Hugo PUC

N3 Informatique
28/04/2021

Rendu projet réseaux

Pour présenter mon projet, je vais séparer mes explications en prenant comme référence le main de mon programme qui nous donne déjà une chronologie logique de la création du qr code.
Mon main se divise en 3 parties, l'encodage de la donnée que l'on veut mettre dans la matrice (le qr code), le remplissage complet de la matrice (la donnée + le reste) et l'application de graphismes sur notre matrice.

Le main sera englobé par un "try catch" pour pouvoir capturer les erreurs éventuelles et les rendre plus lisibles qu'un "null pointer exception" qui viendrait de nulle part.

Encodage de la donnée:

L'encodage de la donnée est la partie la plus intéressante du projet (pour ma part), je vais d'abord vous expliquer ma manière d'encoder la String que l'utilisateur va nous fournir en bit en justifiant mes choix.

```java
public static void main(String[] args)
{
    try{
        // constructeur en prenant que la donnée que l'on veut encoder
        matrice matrice1 = new matrice( data: "pukito");
```

On commence par créer notre objet "matrice" que l'on nomme "matrice1" en lui fournissant comme paramètre une String (ici pukito).

```java
public class matrice {

    private int taille;
    private int [][] matrice;
    private final String data;
    private String binaryData;

    /* Constructeur simple pour donner la donnée à notre objet matrice */

    public matrice(String data)
    {
        System.out.println();
        this.data = data;
    }
```

On a donc un constructeur simple qui va récupérer le String “pukito” et l’assigner à notre objet matrice dans une variable “final” pour que ce String ne soit pas modifiable par la suite.

```
// méthodes pour définir la bonne taille de la matrice
matrice1.FinalDataEncoded();
int taille = matrice1.TailleMatrice();
int [][] m = new int [taille][taille];
matrice1.setTaille(taille);
matrice1.setMatrice(m);
```

Ensuite on va pouvoir jouer avec la String fournie, on lance la méthode “FinalDataEncoded()” pour former notre donnée en bit que l’on mettra dans la matrice.

```
public void FinalDataEncoded() throws DataLengthException {
    binaryData = longueurDonnees() + SplitData() + SplitData();
```

Cette donnée sera composée d’une longueur de donnée totale, de la String encodée 2 fois pour appliquer la redondance.

La longueur de la donnée sera toujours sur 15 bits, la taille de matrice maximale suit la formule suivante:

```
/* Calcule la taille libre dans la matrice pour mettre nos données */

public int calcDonnees(int t) { return (t*t) - (8*8*3) - ((t-16)*2); }
```

177x177 : (Nombre de bits de la matrice au total)
- 8x8x3 : (Les 3 carrés de positionnement)
- (177-16) x 2 : (Les deux lignes pour la taille des pixels)

On expliquera les deux derniers termes plus tard.

Au max on a donc 30815: 111100001101001 (15 bits).

```java
public String longueurDonnees()
{
    new StringBuilder();
    StringBuilder dataLength = new
            StringBuilder(Integer.toBinaryString( i: SplitData().length() * 2));

    if (dataLength.length() != 15)
    {
        while(dataLength.length() != 15)
        {
            dataLength.insert( offset: 0,  i: 0);
        }
    }

    return dataLength.toString();
}
```

La méthode qui retourne la longueur de la donnée va donc prendre la taille de la donnée en bits (reçu avec SplitData()) multipliée par deux puis ajoute des 0 pour qu'on est bien cette longueur de donnée sur 15 bits en permanence.

Maintenant expliquons la grosse méthode "SplitData()" qui contient aussi la méthode "Data()".

On rappelle que le code Ascii de chaque caractère est sur 7 bits, le but va être de prendre chaque lettre de la String et les transformer en bits.
Nous allons utiliser une autre classe pour ce faire: StringConvert.java
On va pouvoir avoir notre "pukito" en bits (donc 6 lettres x 7 bits).
On a aussi forcé le fait que chaque lettre soit sur 7 bits car il y a des cas spéciaux qui peuvent être sur 6 bits si on a pas de chance.

Maintenant que notre "pukito" est encodé en série de 7 bits, on va pouvoir appliquer le code correcteur.

J'ai choisi le code correcteur de Reed-Solomon.
Pour ajouter notre code correcteur, on va découper notre String initiale en morceau de 3.

"pukito" sera coupé en "puk" et "ito".
"pukito1" sera coupé en "puk", "ito" et "1".
"pukito2" sera coupé en "puk", "ito" et "12".

Le code correcteur ajoutera 2 blocs à chaque morceau de notre String.

“puk” + 2 blocs, “ito” + 2 blocs …

Le 1er bloc est l’addition de la valeur décimale en Ascii des lettres de chaque morceau.

“puk” = 112 + 177 + 107

Ce 1er bloc prendra au maximum “zzz” qui est équivalent à “122 + 122 + 122”, cela donne 366, qui peut être codé sur 9 bits.

On va toujours prendre 9 bits pour l’encodage du 1er bloc correcteur.

Le 2eme bloc est l’addition avec la multiplication par coefficient de chaque morceau.

“puk” = 112x1 + 177x2 + 107x3

Le 2ème bloc prendra au maximum 732, qui peut être codé sur 10 bits.

On va aussi forcer le fait que ce 2eme bloc soit sur 10 bits en ajoutant des 0 comme fait précédemment.

Pour résumer, la méthode “Data()” va pouvoir transformer notre String en bits en ajoutant des blocs correcteurs à tous nos morceaux de 3 (et au dernier morceau de 2 ou de 1 cela dépend de la String).
La méthode “SplitData()” va ensuite appeler cette méthode “Data()” en prenant les 3 cas, morceau de taille 3, morceau de taille 2 ou morceau de taille 1 puis renvoyer la donnée finale à la méthode “FinalDataEncoded()”

```
public void FinalDataEncoded() throws DataLengthException {
    binaryData = longueurDonnees() + SplitData() + SplitData();
```

qui va s’occuper de mettre la longueur de la donnée (multiplié par 2) avec notre donnée venant de “SplitData()” deux fois. Rappelons que nous connaissons toutes les tailles de chaque partie de la donnée, quand on voudra lire le qr code, il n’y aura pas de zone de flou.

```
// méthodes pour définir la bonne taille de la matrice
matrice1.FinalDataEncoded();
int taille = matrice1.TailleMatrice();
int [][] m = new int [taille][taille];
matrice1.setTaille(taille);
matrice1.setMatrice(m);
```

Rappelons où on en est, on a réussi à avoir notre donnée encodé en bits avec le code correcteur et de la redondance.
Maintenant que nous avons cette nouvelle String, on pourrait toujours prendre une matrice de taille 177x177, ce n'est pas très malin…
Pourquoi pas adapter la taille de la matrice en fonction de la taille de notre String encodée?

C'est exactement ce qu'on va coder dans la méthode "TailleMatrice()" qui va retourner une taille que l'on va récupérer dans une variable puis on va créer notre matrice (réellement) avec cette taille finale.

On va commencer avec une taille de matrice minimale de 21x21 et on va incrémenter cette taille de 4 à chaque fois.
Ce qui donne 21, 25, 29, 33… On s'arrête à une taille max de 177x177 (rappelez vous c'est la taille qu'on a utilisé pour calculer notre taille de bit max quand on a encodé le String).

Pour savoir la taille que l'on va prendre, on commence à 21 et on compare la taille de notre String encodée au nombre de données que l'on pourrait mettre dans une matrice de 21x21.

```
public int calcDonnees(int t) { return (t*t) - (8*8*3) - ((t-16)*2); }
```

Ce nombre de données est calculé ici (expliqué précédemment).

Tant que la taille de notre String encodée est supérieure à la taille recevable de données selon la formule de "calcDonnees()", on incrémente de 4 cette taille qui est initialement à 21.

```java
public int TailleMatrice() throws DataLengthException {
    int finalTaille = 21;

    while (binaryData.length() > calcDonnees(finalTaille))
    {
        if (finalTaille < 177)
        {
            finalTaille += 4;
        }
        else
        {
            throw new DataLengthException(finalTaille + 4);
        }
    }

    return finalTaille;
}
```

Super ! Maintenant on a une taille de matrice qui va changer en fonction de la taille de la String encodée que l'on prend.

Mais on a un problème plutôt gros…

Que se passerait-il si on prenait une donnée vraiment grande qui dépasserait la taille maximale de 177x177?

Tout le reste du main est inutilisable car notre Méthode "TailleMatrice()" ne renverrait pas de taille finale, hop une "null pointer exception" car toutes les variables de notre objet (la taille, le tableau double qui forme la matrice) ne serait pas remplis donc on ne pourrait pas afficher notre matrice graphiquement…

Et c'est ici qu'on va utiliser le "try catch" qui est censé capturer les erreurs possibles avant de casser tout le programme.

Si on dépasse la taille maximale, on jette une exception avec le "throw new" vert une classe que l'on créé exprès pour gérer cette erreur nommée "DataLengthException.java".

```java
public class DataLengthException extends Exception
{
    public DataLengthException(int t)
    {
        System.out.println("La longueur de votre donnée est trop longue.");
        System.out.println("cela donne une matrice de: "+t+"x"+t+" " +
                "alors que la taille max est de 177x177.");
    }
}
```

Cette classe va avoir une méthode qui prend un entier en paramètre (la taille que l'on a essayé d'utiliser) et va afficher un message personnalisé visible ci-dessus).

Super ! Le code est beaucoup plus solide maintenant et on évite de prendre des erreurs inutiles quand on veut s'amuser à prendre une String énorme.

Remplissage de la matrice

On a fait le plus dur, la donnée est prête à être intégrée dans notre matrice, qu'est ce qu'on va pouvoir mettre dans cette matrice qui formera le qr code?

Trois types de valeur que va prendre la matrice: 0, 1 et 2.

0 et 1 correspondent aux données binaires (blanc et noir usuellement).
le 2 correspondra au vide, dans la plupart des cas on ne pourra pas remplir complètement la partie réservée aux données de la matrice, donc on mettra des 2 à la place de mettre des 0 ou des 1 aléatoirement.

```java
// méthodes pour la construction de la matrice
matrice1.InitMatrice();
matrice1.MotifSensMatrice();
matrice1.PiloteDeTailleMatrice();
matrice1.DonneesMatrice();
matrice1.AffichageMatrice();
```

On va expliquer toutes ces méthodes de construction.

```
public void InitMatrice()
{
    for (int i = 0; i < taille; i++)
    {
        for (int j = 0; j < taille; j++)
        {
            matrice[i][j] = 2;
        }
    }
}
```

Premièrement on initialise la matrice avec une méthode “InitMatrice()” en attribuant des 2 à toutes les cases car on veut être sur d’avoir une matrice “vide”.

```
for (int i = 0; i < 8; i++)
{
    for (int j = 0; j < 8; j++)
    {
        if ((i == 1 && j != 0 && j != 6) || (j == 1 && i != 0 && i != 6) ||
                (i == 5 && j != 0 && j != 6) ||
                (j == 5 && i != 0 && i != 6) || i == 7 || j == 7)
        {
            matrice[i][j] = 0;
        }
        else
        {
            matrice[i][j] = 1;
        }
```

Maintenant on va placer nos 3 carrés dans la matrice qui vont servir à déterminer le sens de la matrice quand on va vouloir la lire.
Ci-dessus on voit la condition qu’on utilise quand on balaye la matrice avec une boucle pour le 1er carré, je vous épargne les deux autres conditions pour les 2 autres carrés qui varient juste au niveau des valeurs de i et j.

```
1 1 1 1 1 1 1 0
1 0 0 0 0 0 1 0
1 0 1 1 1 0 1 0
1 0 1 1 1 0 1 0
1 0 1 1 1 0 1 0
1 0 0 0 0 0 1 0
1 1 1 1 1 1 1 0
0 0 0 0 0 0 0 0
```

On obtient ce carré de 8x8 3 fois dans le les 3 coins de la matrice (en haut à gauche, en haut à droite et en bas à gauche).

Remarquez que la formule 8x8x3 a été utilisée plus tôt pour déterminer l'espace libre qu'on avait pour mettre la donnée dans la matrice.

```
for (int j = 8; j < taille-8; j++)
{
    if (j%2 == 0)
    {
        matrice[6][j] = 1;
    }
    else
    {
        matrice[6][j] = 0;
    }
}
```

Ici on applique la méthode "PiloteDeTailleMatrice()" en mettant une série de 1 et de 0 sur deux lignes de la matrice, ces deux lignes vont reliées les 3 carrés entre eux et on va pouvoir déterminer la taille de chaque pixels quand on va lire le qr code en faisant une fraction toute simple, on prend la taille en cm sur une feuille de notre qr code divisé par le nombre de pixels qui alternent entre 1 et 0 sur la feuille (il faut aussi enlever les carrés qui déterminent la position dans la matrice).

Rappelez vous aussi qu'on a utilisé la formule (taille - 16) x 2 pour enlever les lignes de pilote de taille quand on voulait calculer l'espace libre pour mettre nos données dans la matrice.

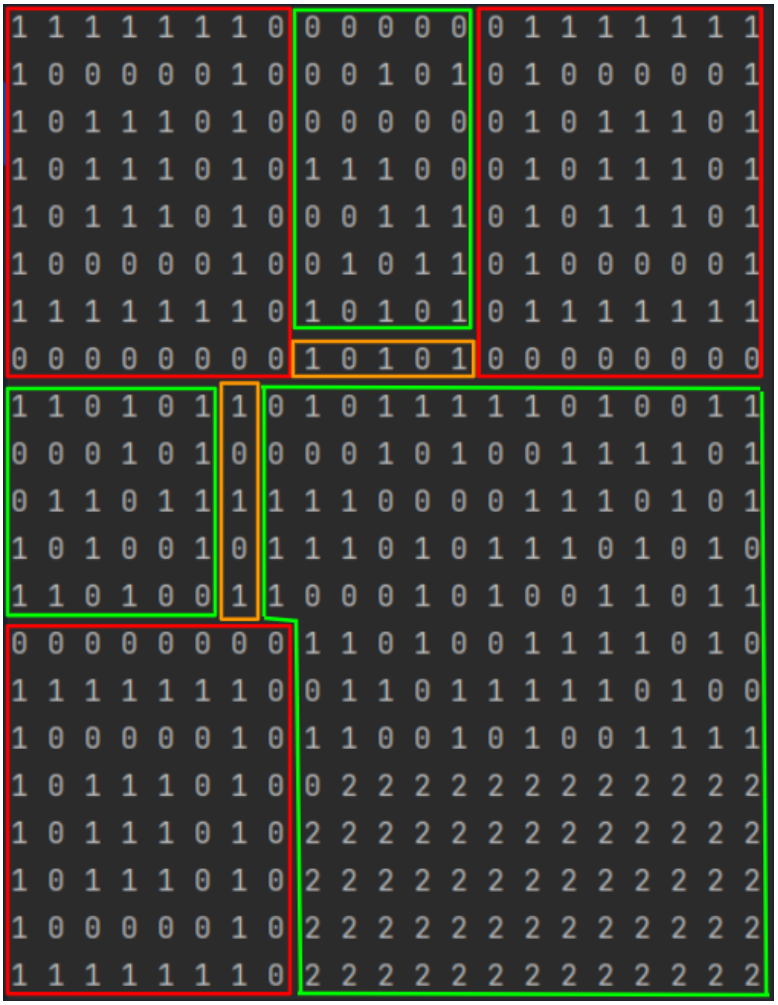
```
public void DonneesMatrice()
{
    int cpt = 0;

    // Bloc de données milieu haut
    for (int i = 0; i < 8; i++)
    {
        for (int j = 8; j < taille-8; j++)
        {
            if (i != 6)
            {
                if (cpt < binaryData.length())
                {
                    matrice[i][j] = binaryData.charAt(cpt) - 48;
                    cpt++;
                }
                else
                {
                    matrice[i][j] = 2;
                }
            }
        }
    }
```

Maintenant que notre matrice est prête, on va pouvoir mettre notre String encodée !

Pour ce faire, on se place dans les cases libres de notre matrice et on utilise les bits de notre String encodée 1 à 1 en parcourant cette String.
Tant que le cpt ne dépasse pas la taille de notre String, on peut continuer à placer les bits de la String 1 à 1, quand la String a fini d'être parcouru, on fini de remplir la matrice par des 2 pour expliciter que l'on met du vide pour compléter cette espace inutilisé. On oublie pas d'enlever le code Asciis (-48) à chaque fois sinon on aura des 48 et 49 au lieu d'avoir des 0 et 1.

```java
public void AffichageMatrice()
{
    for (int i=0; i<taille; i++)
    {
        for (int j=0; j<taille; j++)
        {
            System.out.print(matrice[i][j]);
            System.out.print(" ");
        }
        System.out.println();
    }
}
```

Et voilà, notre matrice a fini d'être remplie, on va pouvoir l'afficher !



En rouge, les 3 carrés de positionnement, en orange les 2 lignes pour la taille des pixels de la matrice et en vert les données.

Le graphisme

On va pouvoir faire s'amuser avec les couleurs que l'on va donner à notre qr code, maintenant qu'on a notre matrice remplie de tout ce qu'il nous faut, c'est simple, il suffit de mettre trois conditions en appliquant une couleur pour le 0, une couleur pour le 1 et une couleur pour le 2 qui représente le vide.

```
// graphisme
int [][] matriceCouleur;
matriceCouleur = matrice1.getMatrice();
int w = 525;
int h = 525;

JFrame f = new JFrame();
Drawing d = new Drawing(w, taille,matriceCouleur);
f.setSize(w,h);
f.setTitle("Qr code");
f.add(d);
f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
f.setVisible(true);
```

On prépare ce qui nous faut en chargeant les librairies qui utilisent du graphisme et on va définir une largeur et une hauteur.

Après cela on va créer la classe "Drawing.java" pour s'amuser à mettre un peu de couleur et régler la taille des pixels.

```
public class Drawing extends JComponent
{
    private final int taille;
    private final int [][] matrice;
    private final int taillePixel;
```

Cette classe va prendre une taille, une matrice et une taille de pixel.

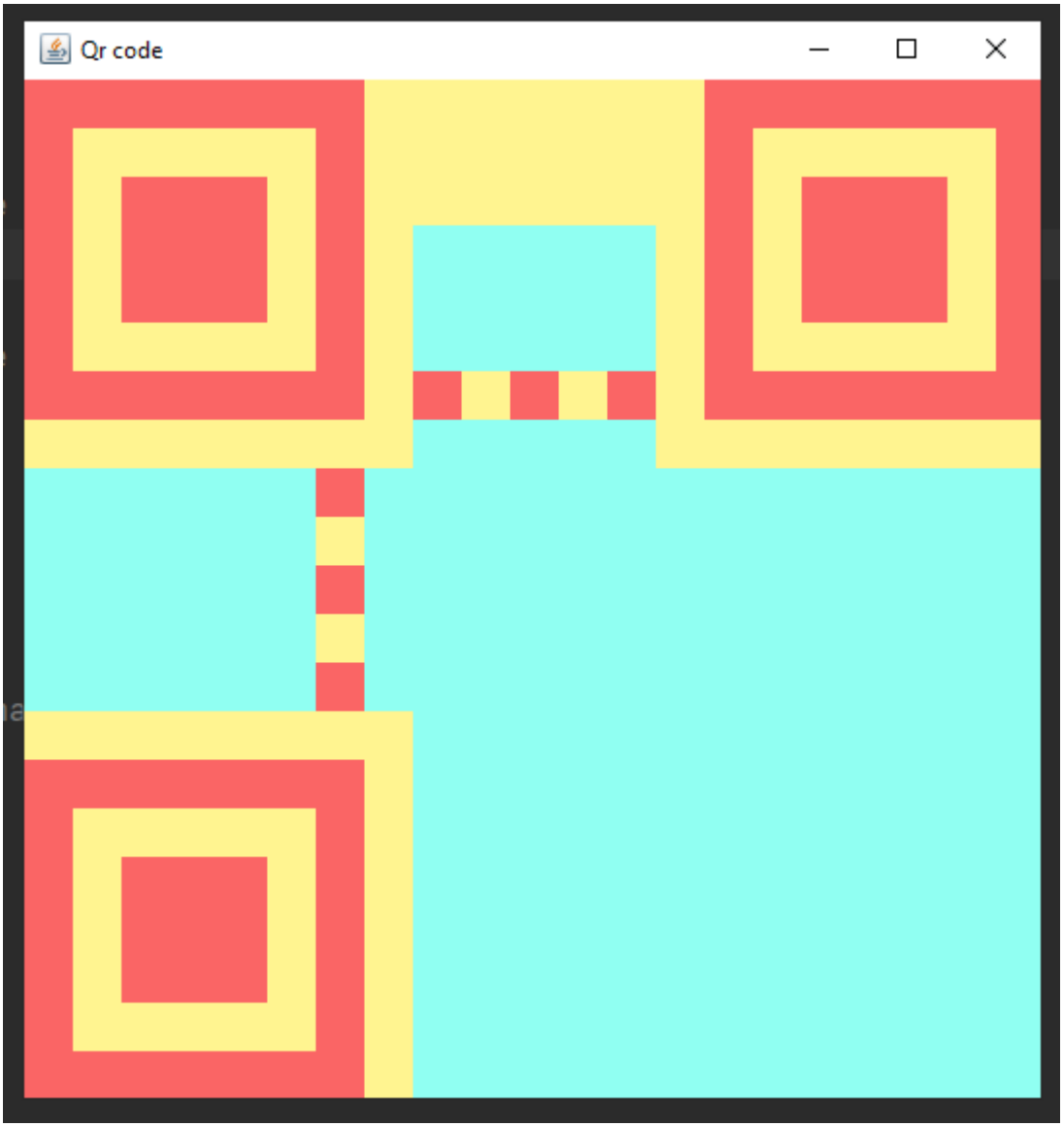
```java
public Drawing(int w, int t, int[][] m)
{
    taille = t;
    matrice = m;
    taillePixel = w /taille;
}
```

Le constructeur va récupérer la taille, la matrice puis va prendre pour taille de pixel la taille de la fenêtre (525) divisé par la taille de la matrice, plus la taille de la matrice est grande, plus petit les pixels seront pour pouvoir faire tenir tout le qr code dans notre fenêtre, si on n'avait pas fait cela, on aurait pu afficher correctement qu'une taille spécifique de matrice.

```java
protected void paintComponent(Graphics g)
{
    Graphics2D g2d = (Graphics2D) g;

    Rectangle2D.Double[][] r3 = new Rectangle2D.Double[taille][taille];

    for (int i=0; i<taille; i++)
    {
        for (int j=0; j<taille; j++)
        {
            if(matrice[i][j]==0)
            {
                g2d.setColor(new Color( r: 255,  g: 244,  b: 144));
                r3[i][j] = new Rectangle2D.Double( x: j*taillePixel, y: i*taillePixel,
                        taillePixel,taillePixel);
                g2d.fill(r3[i][j]);
            }
```

Puis avec la méthode "paintComponent()", on va créer un rectangle (qui est en fait un carré) à chaque fois qu'on rencontre un 0, un 1 ou un 2, on va donner une couleur par chiffre et on va positionner ces carrés en fonction de i et j qui parcours notre matrice, i et j seront multiplié par la variable "taillePixel" calculé en fonction de la taille de la matrice précédemment.

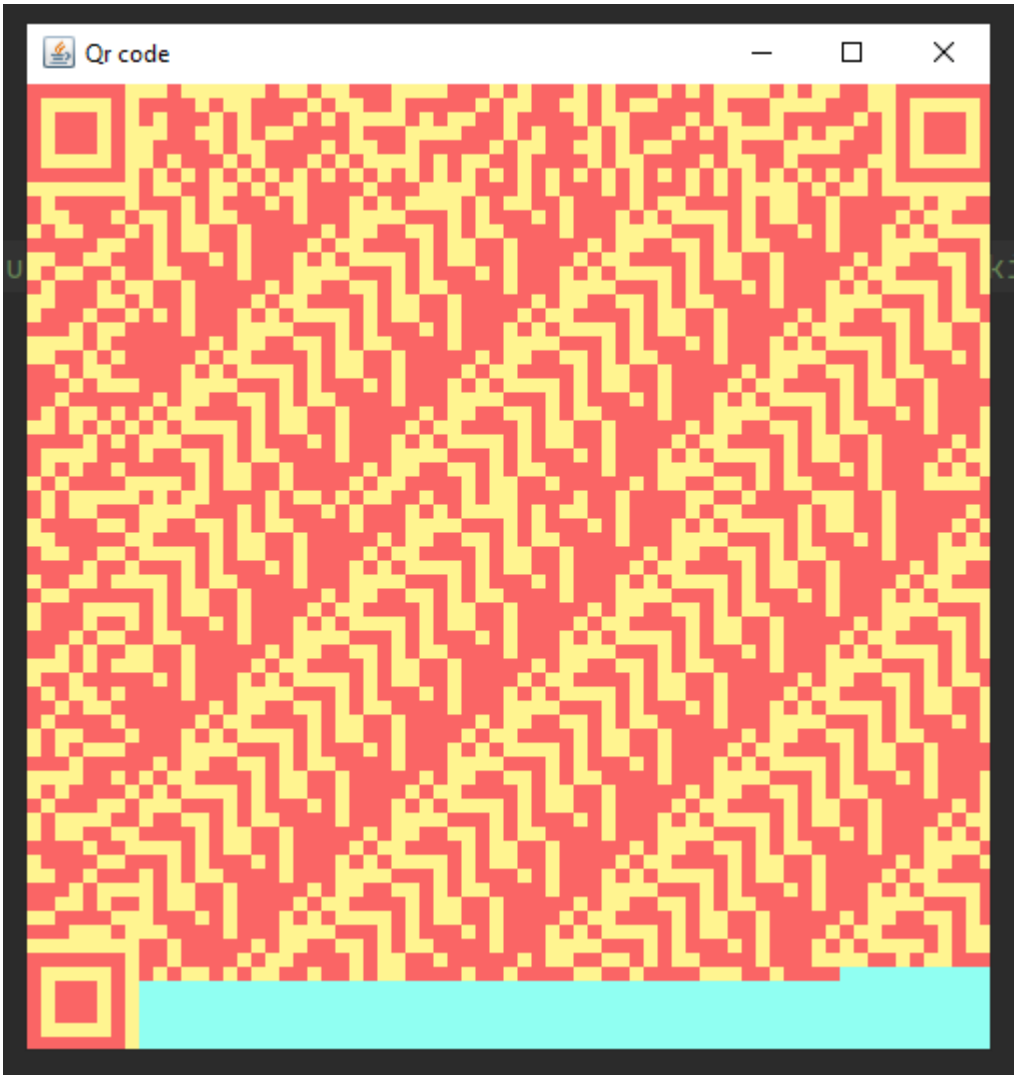
Et voilà le résultat, on a exactement la même chose mais avec des couleurs. Ici on a pris comme String initiale “pukito”.



Si on ne met rien dans la String initiale, on va avoir que les 15 premiers bits de données remplis à 0 car cela correspond à la longueur de la donnée que l'on mettra systématiquement sur 15 bits, ici la longueur de la donnée est de 0 donc 15 bits à 0.

```
32015
La longueur de votre donnée est trop longue.
cela donne une matrice de: 181x181 alors que la taille max est de 177x177.

Process finished with exit code 0
```

Quand on prend une chaîne de caractère trop longue, l'erreur est bien affichée.



Qr code en prenant plein de “pukito” à la suite pour notre String initiale.

J’espère que la présentation de mon projet vous aura plu, grâce à ce projet j’ai pu apprendre beaucoup de choses en java, le but étant de m’améliorer dans ce langage, la possibilité de choisir le langage de programmation était une bonne opportunité.
J’ai aussi pu m’amuser à mettre toutes les versions de mon code à jour dès que j’avançais sur le projet avec GitHub, je vous laisse le lien de mon GitHub si jamais vous voulez suivre mes progrès sur mes autres projets !
https://github.com/hugopukito?tab=repositories