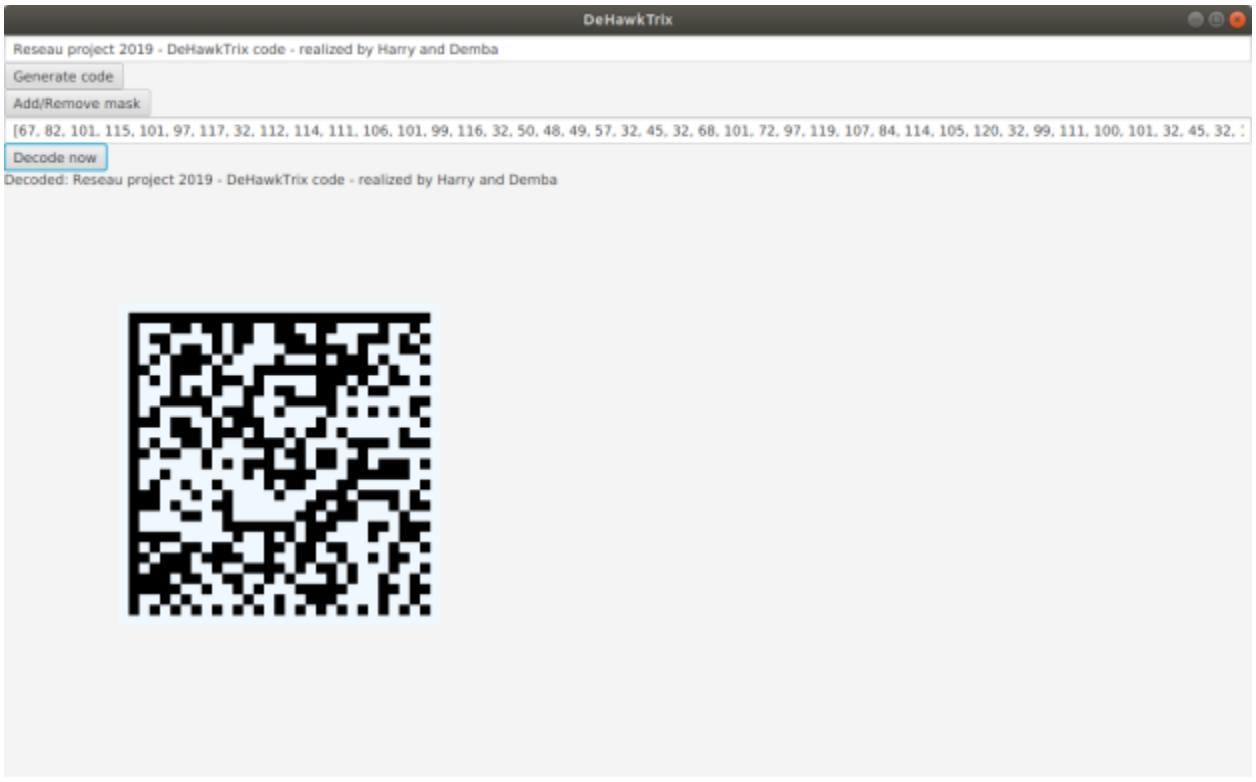
Auteurs: CISSE Demba
JANDU Harry

RAPPORT DeHawkTrix Code

Licence Informatique 3ème année **2018-2019**

Introduction

DeHawkTrix est une barcode sous forme de matrice (en 2D) qu'on peut imprimer en utilisant des rectangles. Les données sont représentées sous forme d'une série en utilisant des couleurs obscure et visible. Il existe six tailles différentes de la matrices qui seront discutés ci-dessous. Il faut noter que la taille maximum de caractère qu'on puisse coder dans un DeHawkTrix est 230 caractères (en excluant 11 caractères pour la correction des erreurs et 1 caractère pour stocker la taille du message). Un exemple d'un DeHawkTrix est donné ci-dessous:



Barcode genere par le DeHawkTrix avec un mask

Structure de notre barcode

La structure est à peu près comme un DataMatrix, le Finder Pattern qui est donné par les dotted lines. Il suit un format de 101010..ainsi de suite en allant de gauche à droite en bas et de haut en bas à droite. Il est utile pour donner l'orientation de notre barcode. Il y aura aucun donnees qui vont intervenir dans le Finder Pattern. Ensuite, la ligne solide commençant d'en bas a gauche allant en haut jusqu à droite est les bordures de notre barcode. Le scanner l'utilisera pour savoir la taille de la matrice. L'image ci-dessus est la version 3 avec taille 31 x 31 qui peut contenir 86 caractères. Le Finder Pattern et les bordures sont contenu dans 30x30, ce qui veut dire que les 1x1 qui reste en dehors est utilisé pour le Quiet Zone (une espace vide pour séparer le code d'autre contenu dans une page avec des données).

Comment les données sont codées

Une cellule est représentée par un bit (0 ou 1). La taille de cellule dans l'image ci-dessus est 10px et ca peut être modifiée dans le code. Quand le bar code est dessine, les zéros sont representee par un couleur qui reflète la lumière (comme blanc) et les 1 sont representee par un couleur qui absorbe la lumière (noir). Pour remplir la matrice, les données sont d'abord converti en binaire et puis ils sont entrées dans la matrice. La suite dans laquelle les données seront représentés dans tous les DeHawkTrix barcodes.

taille_du_message + message + error_correction_code
(1 byte + n bytes + 11 bytes)

Après que les données sont insérés dans le barcode, on utilise un mask pour ne pas avoir trop d'espaces vides ou blancs. Le mask est calculé par la fonction suivante

state = 0 ou 1;
f(x, y) = (((float) i * 7.8) + ((float) j * 2.3)) % 7;
$\forall\ i, j\ \varepsilon\ [3,\ taille\,Matrice\ -\ 2]$
si f(x, y) == (2, 3, ou 5) => matrice[i][j] = !state

On inverse l'état de la cellule si la condition est vérifiée. Donc si la cellule était de couleur blanc, elle devient noir, et vice versa.

Tables de capacites de DeHawkTrix

Version	Taille	Region de donnees	Donnees	Corrections erreurs	Capacite totale
1	21x21	18x18	29	11	40
2	27x27	24x24	61	11	72
3	31x31	28x28	87	11	98
4	35x35	32x32	117	11	128
5	43x43	40x40	189	11	200
6	47x47	44x44	231	11	242

Remarque: Le DeHawkTrix ne supporte que les caractères alphanumériques.

Representation des polynomes dans notre code

Avant de passer à la correction d'erreur, il faut savoir comment les polynômes sont représentés dans le code. On a choisi de représenter les polynômes par l'ordre décroissante dans un tableau d'entiers. Par exemple, si on a le polynôme

$$x^5 + x^2 + 1$$

Ca sera représenté dans un tableau

$$[1,\ 0,\ 0,\ 1,\ 0,\ 1]$$

Les éléments sont les coefficients du polynômes et les indices sont le degré.

Encodage de données

Un message passe au programme sera interprété sous forme d'un polynôme dont les coefficients seront les codes ascii du caractère. Par exemple, le message

$Hello$

sera represente par le polynome

$$72x^4 + 101x^3 + 108x^2 + 108x + 111$$

Correction d'erreurs

Pour la correction d'erreurs, on s'est inspiré de la correction de Reed-Solomon. On ne va pas trop détailler mais il est important d'être familier avec les *Corps Fini*[1]. Nous avons choisi de créer un corps fini de $2^8 = 256$. Par conséquent, ça ne sera pas possible de coder plus de 255 caractères. Comme le Reed-Solomon utilise un générateur de polynôme, on a utilisé le *polynôme irréductible*[2] (on peut le visualiser comme un nombre premier mais pour les polynômes)

$$285 = 0b100011101 = x^8 + x^4 + x^3 + x^2 + 1$$

Pour générer le Galois Field et pour ne pas répéter des nombres, on va faire multiplier par 2 et prendre le modulo avec le polynôme irréductible.
Par exemple, on peut avoir ce suite:

$1,\ 2,\ 4,\ 8,\ 16,\ 32,\ 64,\ 128,\ 29,\ 58,\ 116,\ ...$

[1] https://fr.wikipedia.org/wiki/Corps_fini
[2] https://fr.wikipedia.org/wiki/Polyn%C3%B4me_irr%C3%A9ductible

Comme on peut remarquer, après avoir fait $128 * 2$, on a trouve 29. L'idée est de faire un xor avec 285.

$$256 \; xor \; 285 \; = \; 29$$

Et chaque fois qu'on dépasse 256, on fait le modulo encore. Les nombres ne font que répéter quand l'itération arrive a 255.

Ensuite, on utilise un générateur de polynôme avec `n` bits de vérification d'erreurs et le polynôme qui est généré, on va diviser le message par ce polynôme comme on fait pour les codes CRC.

La partie d'encodage est facile, le remnant de la division est concaténé à la fin du message et ca sera les codes de corrections.

Exemple de encoder le message "Hello".

[5, 72, 101, 108, 108, 111, 37, 162, 135, 224, 13, 39, 26, 175, 111, 63, 104]

`5` est la taille du message suivi par le polynôme `Hello` suivi par 11 caractères de correction du codes.

Important: Ca ne sera pas possible de récupérer un message si les codes d'erreurs sont détruites.

Decodage
Pour décoder, il faut d'abord créer le "syndrome polynomial". Ce polynôme sera comprises de zéro si il n'y a aucune erreur, sinon il renverra un polynôme non-nul. Le deuxieme etape est de chercher ou l'erreur se trouve en utilisant l'algorithme de *Berlekamp-Massey*[3]. Cette fonction va renvoyer un polynôme qui sera utilisé pour chercher la position d'erreur dans le message code.

Un exemple en utilisant le message "Hello".
Supposons que l'indice 1 et 3 étaient modifie.

[5, 64, 101, 0, 108, 111, 37, 162, 135, 224, 13, 39, 26, 175, 111, 63, 104]

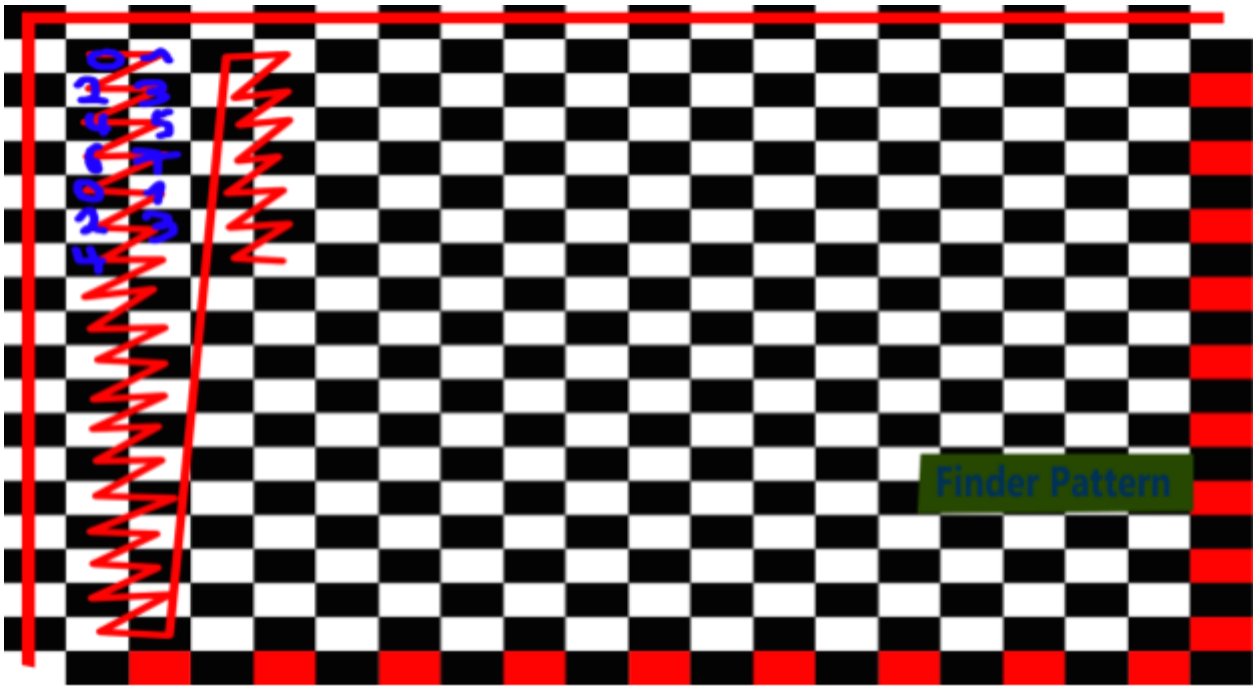
Apres avoir calcule le "syndrome polynomial", on obtient
[0, 100, 48, 82, 8, 130, 193, 124, 97, 67, 160, 161]
qui est non-nul et donc il y a eu des erreurs.
Ensuite, on cherche a localiser les erreurs

[3] *http://www.ijceronline.com/papers/Vol2_issue8/R02801270130.pdf*

[24, 161, 1] - en utilisant Berlekamp-Massey (on pourrait aussi utiliser Extended Euclidean Algorithm)
Finalement, il faut trouver les positions dans laquelle l'erreur s'est produit
[3, 1]

```
input: [ 5 72 101 108 108 111 37 162 135 224 13 39 26 175 111 63 104 ]
Syndrome polynomial: [0, 100, 48, 82, 8, 130, 193, 124, 97, 67, 160, 161]
Localization polynomial: [24, 161, 1]
errors' positions: [3, 1]
```

L'exemple de Hello execute par DeHawkTrix



L'image montre comment les bits sont inseres dans la matrice, l'insertion commence de haut a gauche et on insere 2 bits dans 2 colonnes a la fois.

Comme montré dans l'image, on insère les données en 2 colonnes à la fois et on saute la ligne et tant qu'on n'a pas fini d'insérer dans toutes les lignes, on bouge pas vers la colonne suivante.

Fonctionnalités du programme
Le programme est GUI et l'utilisateur peut entrer le message qu'il voudrait visualise dans le DeHawktrix dans le premiere champ au dessus du bouton "Generate code". Une fois le message entrer, il faut cliquer sur le bouton "Generate code" et ca va afficher le DeHawkTrix bar

code. De plus, l'utilisateur peut visualiser l'image sans le mask. Voici un exemple de deux images avec le même message - une qui utilise les masque et l'autre qui n'utilise pas.

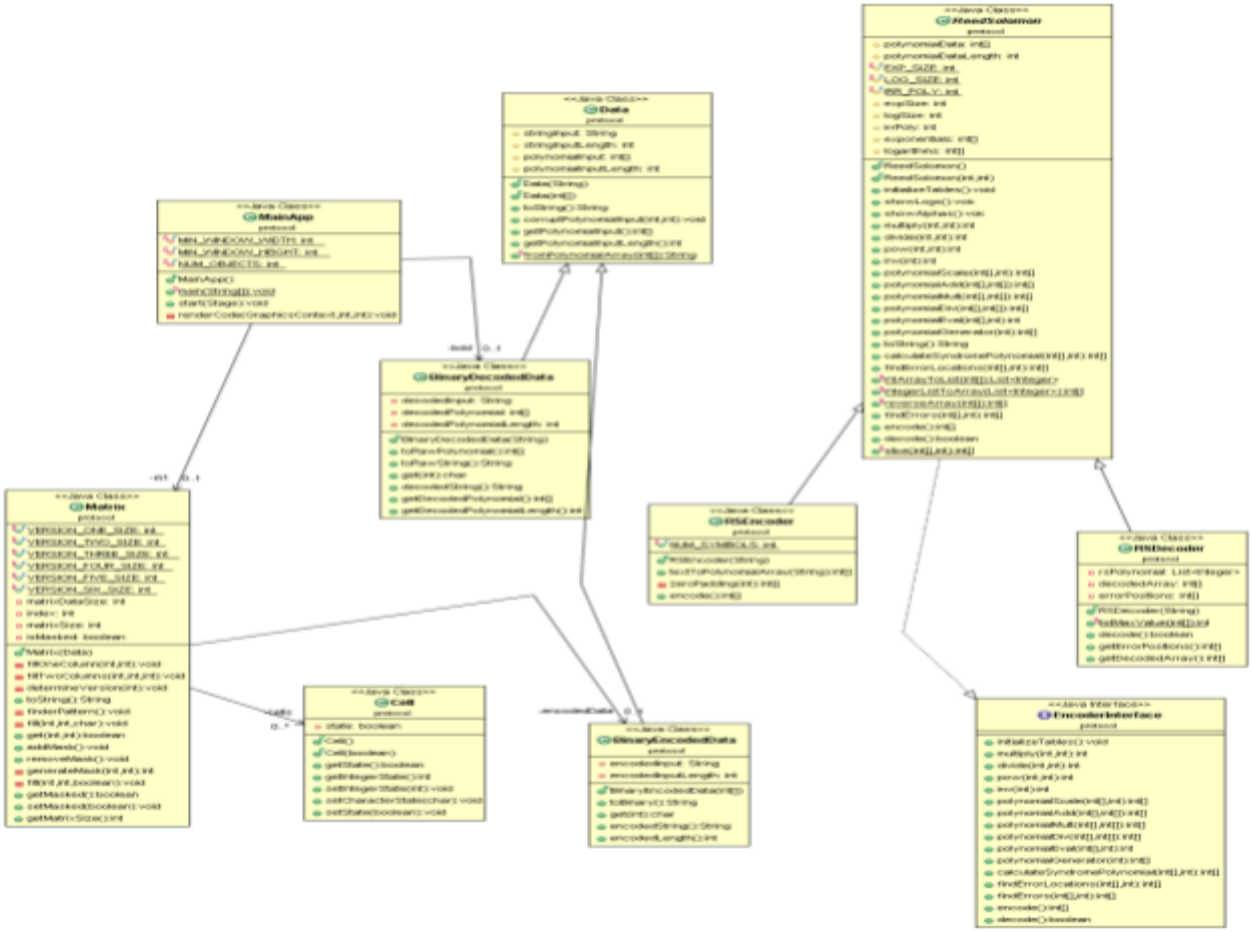
Image avec mask

Image sans mask

La matrice de données peut être vu dans le terminal lors d'exécution de la programme accompagné par des messages descriptifs. Finalement, l'utilisateur peut changer le message code qui s'affiche dans le deuxième champ. Apres avoir change, le programme essaie de déterminer si le message est intacte, si ce n'est pas le cas, il affiche les positions dans lesquelles l'erreur s'est produit. Il est impératif de noter qu'il ne faut pas changer les codes d'erreurs ainsi que le code ne corrige ni les erreurs ni les erasures.
Par contre, il y a la possibilité d'ajouter le code de correction des erreurs mais on n'a pas pu l'implémenter à cause des contraintes des heures. A la fin, le code reste complet avec beaucoup d'évolutions possibles. C'est aussi possible de incrémenter le nombre maximum de caractères qu'on puisse coder en changeant la puissance de l'espace Galois mais il faut obligatoirement changer le polynôme irréductible.

Remarque: Il existe toujours un ou plusieurs polynômes irréductibles dans $[2^n, 2^{n+1}]$ ou n est un entier positif.

UML de notre projet



Conclusion

Ce projet n'est qu'un début dans le monde des barcodes de deux dimensions. On peut ajouter d'autres fonctionnalités comme coder un logo dans le barcode ou encore améliorer le Finder Pattern pour mieux utiliser l'espace des cellules et ajouter la possibilité de coder des caractères non-ascii.

Pour le futur, on réfléchit à utiliser les couleurs différents pour l'encodage et ca peut etre une amélioration au point qu'on n'aura pas besoin d'utiliser un Finder Pattern.

Bibliographie

1. https://www.academia.edu/31243287/Reed_Solomon_Encoding_Simplified_Explanation_for_Programmers
2. https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-451-principles-of-digital-communication-ii-spring-2005/video-lectures/lecture-11-reed-solomon-codes/
3. https://www.youtube.com/watch?v=x1v2tX4_dkQ
4. https://www.gs1ca.org/pages/n/standards/GS1_DataMatrix_Introduction_and_technical_overview_v.pdf
5. https://ent.uca.fr/moodle/pluginfile.php/692201/mod_resource/content/1/indications_TP_reseau_II.pdf