Les codes RR, comment que c'est-ti qu'ils marchent-ils donc?

Tiens mais c'est vrai, c'est une bien bonne question

Michael PAPER

December 30, 2024

Alors pour rappel, on parle de ces trucs

Premier exemple

Figure 1: Un joli p'tit code RR (allez vas-y, fais-toi plaisir, scanne-le)

Un autre exemple

Figure 2: Un code RR un peu moins p'tit mais avec un p'tit truc en plus

Encore un autre exemple

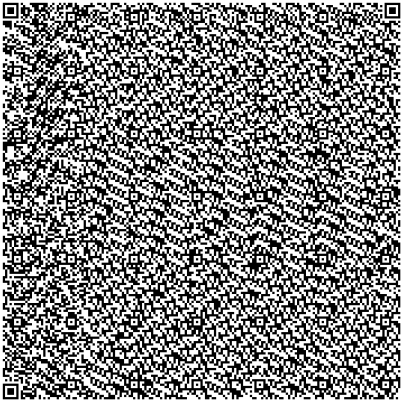
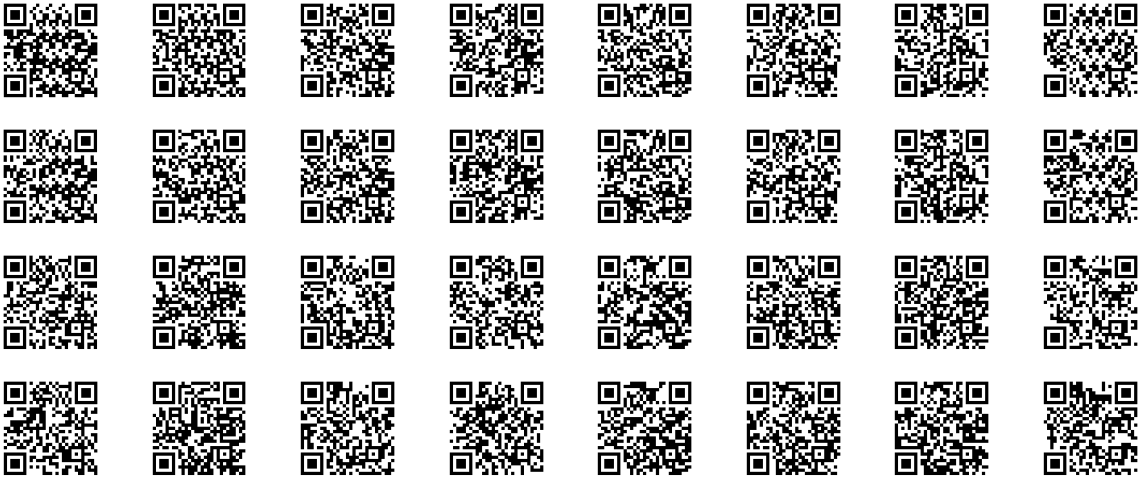


Figure 3: Un gros gros code tout gros

Encore d'autres exemples!



Allez hop on est partis on va en faire un

On va tirer au sort le contenu du code RR

On va tirer au sort le contenu du code RR

En fait nan lol

On va tirer au sort le contenu du code RR

En fait nan lol

On va faire

`https://michaelpaper.xyz`

On va tirer au sort le contenu du code RR

En fait nan lol

On va faire

`https://michaelpaper.xyz`

C'est moi qui choisis

On va tirer au sort le contenu du code RR

En fait nan lol

On va faire

`https://michaelpaper.xyz`

C'est moi qui choisis

Si t'es pas content bah c'est pareil

On va tirer au sort le contenu du code RR

En fait nan lol

On va faire

`https://michaelpaper.xyz`

C'est moi qui choisis

Si t'es pas content bah c'est pareil
Allez c'est parti !

D'abord il faut choisir des trucs

On doit choisir la taille du code RR

- Il y a des "versions" entre 1 et 40
- Version n a une largeur de $4n + 17$
- On va faire version 4
- C'est moi qui choisis
- Donc 33×33
- Tu vas faire quoi

On doit choisir le niveau de redondance

- $+$ de redondance $\rightarrow$ $-$ de contenu
- 4 niveaux de redondance possibles
- On va prendre le max
- C'est pas toi qui choisis
- C'est moi qui choisis
- Viens te battre si t'es pas content

Ensuite il faut encoder le contenu !

Plusieurs types d'encodage:

1. Kanji
2. Binaire (en gros, ASCII)
3. Juste des chiffres
4. Des chiffres et des lettres

- On peut faire plusieurs segments dans un seul symbole
  - Mais flemme.
- Moi je comprends pas les kanji.
- L'ASCII c'est tout ennuyeux.
- Y a pas que des chiffres

Ensuite il faut encoder le contenu !

Plusieurs types d'encodage:

1. Kanji
2. Binaire (en gros, ASCII)
3. Juste des chiffres
4. Des chiffres et des lettres

- On peut faire plusieurs segments dans un seul symbole
  - Mais flemme.
- Moi je comprends pas les kanji.
- L'ASCII c'est tout ennuyeux.
- Y a pas que des chiffres
- Alors on va faire (4) !

Chaque chiffre/lettre/ponctuation a une valeur (table 5 dans la spec)

'0'	→	0	'D'	→	13	'Q'	→	26	'*'	→	39
'1'	→	1	'E'	→	14	'R'	→	27	'+'	→	40
'2'	→	2	'F'	→	15	'S'	→	28	'-'	→	41
'3'	→	3	'G'	→	16	'T'	→	29	'.'	→	42
'4'	→	4	'H'	→	17	'U'	→	30	'/'	→	43
'5'	→	5	'I'	→	18	'V'	→	31	':'	→	44
'6'	→	6	'J'	→	19	'W'	→	32			
'7'	→	7	'K'	→	20	'X'	→	33			
'8'	→	8	'L'	→	21	'Y'	→	34			
'9'	→	9	'M'	→	22	'Z'	→	35			
'A'	→	10	'N'	→	23	' '	→	36			
'B'	→	11	'O'	→	24	'$'	→	37			
'C'	→	12	'P'	→	25	'%'	→	38			

Calcul conversion complexe

"HT"	→	table5['H'] * 45 + table5['T']	→	17 * 45 + 29	→	794
"TP"	→	table5['T'] * 45 + table5['P']	→	29 * 45 + 25	→	1330
"S:"	→	table5['S'] * 45 + table5[':']	→	28 * 45 + 44	→	1304
"//"	→	table5['/'] * 45 + table5['/']	→	43 * 45 + 43	→	1978
"MI"	→	table5['M'] * 45 + table5['I']	→	22 * 45 + 18	→	1008
"CH"	→	table5['C'] * 45 + table5['C']	→	12 * 45 + 17	→	557
"AE"	→	table5['A'] * 45 + table5['E']	→	10 * 45 + 14	→	464
"LP"	→	table5['L'] * 45 + table5['P']	→	21 * 45 + 25	→	970
"AP"	→	table5['A'] * 45 + table5['P']	→	10 * 45 + 25	→	475
"ER"	→	table5['E'] * 45 + table5['R']	→	14 * 45 + 27	→	657
".X"	→	table5['.'] * 45 + table5['X']	→	42 * 45 + 33	→	1923
"YZ"	→	table5['Y'] * 45 + table5['Z']	→	34 * 45 + 35	→	1565

On en fait des 1 et des 0

Chaque paire tient sur 11 bits.

```
794 ++ 1330 ++ 1304 ++ 1978 ++ 1008 ++ 557 ++ 464 ++ 970 ++ 475 ++ 657
++ 1923 ++ 1565
=
01100011010 ++ 10100110010 ++ 10100011000 ++ 11110111010 ++
01111110000 ++ 01000101101 ++ 00111010000 ++ 01111001010 ++
00111011011 ++ 01010010001 ++ 11110000011 ++ 11000011101
=
01100011010101001100101010001100011110111010011111100000100010110
10011101000001111001010001110110110101001000111110000011110000111101
```

On ajoute des p'tits bidules un peu utiles

- Avant:
  - Le mode d'encodage "des chiffres et des lettres": 0010
  - La taille de ce qu'on encode: 24 $\rightarrow$ 000011000
- Ensuite:
  - **TERMINATOR**: 0000
  - Du padding (en bits): 000
  - Du padding (en octets): 11101100 00010001

0010 000011000 01100011010101001100101010001100011110111010011111100000100010110100111010000011110010100011101101101010010001111100001111101 0000 000 11101100 00010001 11101100 00010001 11101100 00010001 11101100 00010001 11101100 00010001 11101100 00010001 11101100

On a tous les bits !

001000001100001100011010101001100101010001100011110111010011111100000100
010110100111010000011110010100011101101101010010001111100000111100001110
100000001110110000010001111011000001000111101100000100011110110000010001
111011000001000111101100000100011110110000010001111011000001000111101100

On divise le contenu en blocs

00100000110000110001101010100110010101000110001111011101001111110000010
01011010011101000001111001010001110110110101001000111110000011110000111
10000000111011000001000111101100000100011110110000010001111011000001000
11101100000100011110110000010001111011000001000111101100000100011110110

Codes correcteurs d'erreurs de Salomom Reed

- C'est super rigolo
- Mais c'est pas trivial
- Y a des divisions de polynomes et tout le bazar
- Alors on va juste partir du principe qu'on sait les calculer

On calcule les codes correcteurs d'erreurs pour chaque bloc

001000001100001100011010101001100101010001100011110111010011111100000100
101000100001110111001110011010100100111110011001110110101001001110111010
00111101111100010100100001110100111010010110111111000110

010110100111010000011110010100011101101101010010001111100000111100001110
000110100110100100100010011101000010100000000101101011100000000001111001
11111110001110111000000100010100111111001011001000010110

100000001110110000010001111011000001000111101100000100011110110000010001
111000010000000101100100011001100011111001101101111101111110011001110000
01011110100000101101001010000001111011111000111110111111

111011000001000111101100000100011110110000010001111011000001000111101100
011101000110000010000011000101010011101010000110010000000110111001000011
11011100111011010011010010110010101010011101111101111110

On bouge un peu les bidules

001000000101101010000000111011001100001101110100111011000001000100011010
000111100001000111101100101001100101000111101100000100010101010011011011
000100011110110001100011010100101110110000010001110111010011111000010001
111011000011111100001111111011000001000100000100000011100001000111101100
101000100001101011100001011101000001110101101001000000010110000011001110
001000100110010010000011011010100111010001100110000101010100111100101000
001111100011101010011001000001010110110110000110110110101010111011110111
010000001001001100000000111001100110111010111010011110010111000001000011
001111011111111001011110110111001111000100111011100000101110110101001000
100000011101001000110100011101000001010010000001101100101110100111111100
111011111010100101101111101100101000111111011111110001100001011010111111
01111110

Ok maintenant on se lance et on fait un dessin

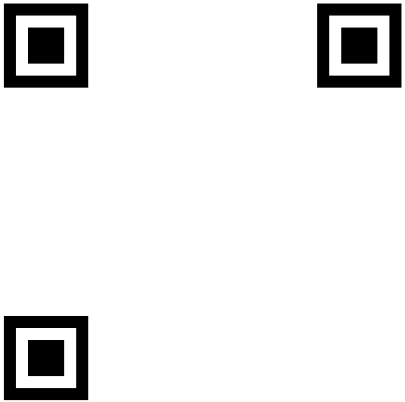


Figure 4: C'est comme un puzzle, faut toujours commencer par les coins

Dessine-moi un mouton

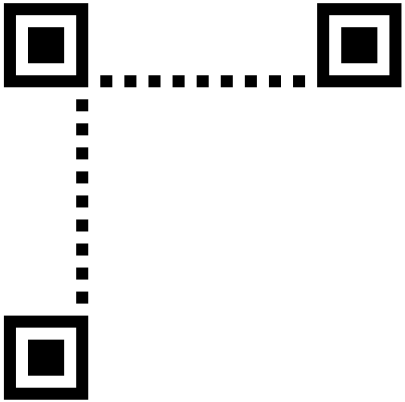


Figure 5: Ensuite faut faire les bords

Dessine-moi un mouton qui louche

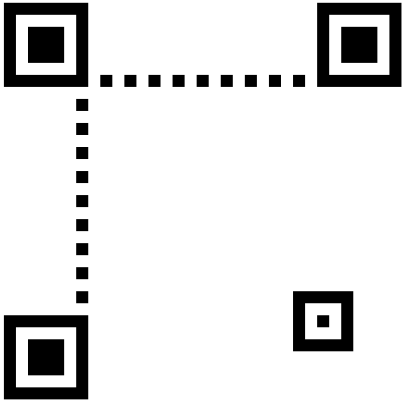


Figure 6: En suivant la table E1 de la spec pour ajouter les yeux qui louchent

On met enfin des infos sur le code RR

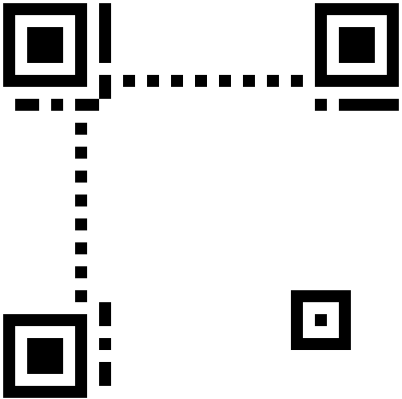


Figure 7: On indique le niveau de correction d'erreur et le masking pattern, avec de la redondance

Maintenant on fait un gros zigzag sur les pixels libres

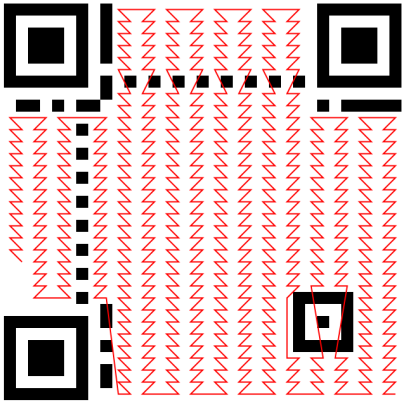


Figure 8: Le nom scientifique de ce type de zigzag existe (je m'en souviens plus)

Et tadam !

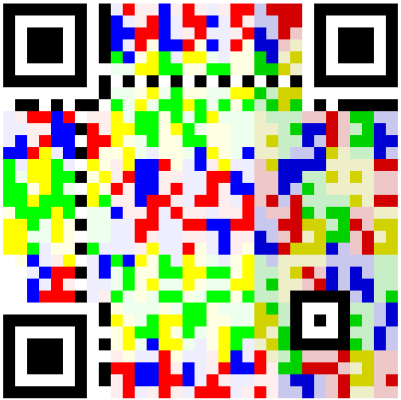


Figure 9: Tadam en couleurs !

Figure 10: Tadam en noir et blanc !

Tadam ?

Tadam ?
Tadam !?

Tadam ?
Tadam !?
Pourquoi il reste des diapos ?

Tadam ?
Tadam !?
Pourquoi il reste des diapos ?
Laissez-nous sortir !

Il faut encore choisir et XORer tout ce bordel avec un masking pattern

P_0

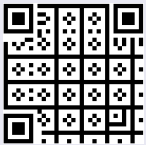
P_1

P_2

P_3

P_4

P_5



P_6

P_7

Vrai tadam !!!

P_0

P_1

P_2

P_3

P_4

P_5

P_6

P_7

Conclusion

Les codes RR c'est RIGOLO !

Les codes RR c'est RIGOLO !
Bisous