UNIVERSITÉ Clermont Auvergne

Théo NICOLAS

RESEAUX II

Protocole graphique

RAPPORT

UNIVERSITÉ Clermont Auvergne

SOMMAIRE

INTRODUCTION

Sujet :

Le sujet du TP était de concevoir un protocole de communication "graphique". Notre protocole devait contenir différentes informations tel que le message à transmettre, ainsi que divers autres informations pouvant servir à détecter ou corriger les erreurs.

Idée :

Dès le début, j'avais pour ambition de concevoir un protocole original qui se distingue totalement du QR code. Pour y parvenir, j'ai décidé de ne pas simplement utiliser une matrice carrée conventionnelle, mais plutôt de m'appuyer sur des cercles. C'est ainsi qu'est venue l'idée d'utiliser les 360 degrés du cercle pour coder le message. Dans les paragraphes suivants, je vais expliquer comment j'ai réalisé cette méthode de codage sur le cercle, ainsi que la façon dont j'ai mis en place la détection et la correction d'erreurs pour assurer une lecture précise des données encodées.

PRINCIPE DE BASE

Dans cette partie, je vais expliquer les principes de base de mon protocole graphique. J'ai décidé de me concentrer sur le codage de caractères ASCII pour assurer une large possibilité de message. Je vais également expliqué comment j'ai réalisé le codage de caractères sur un cercle divisé en 17 parties égales chacune correspondantes à un code Hexadécimal. Cette approche m'a permis de réduire considérablement le nombre de caractères nécessaires pour encoder les données, en comparaison avec du binaire.

L'HEXADÉCIMAL

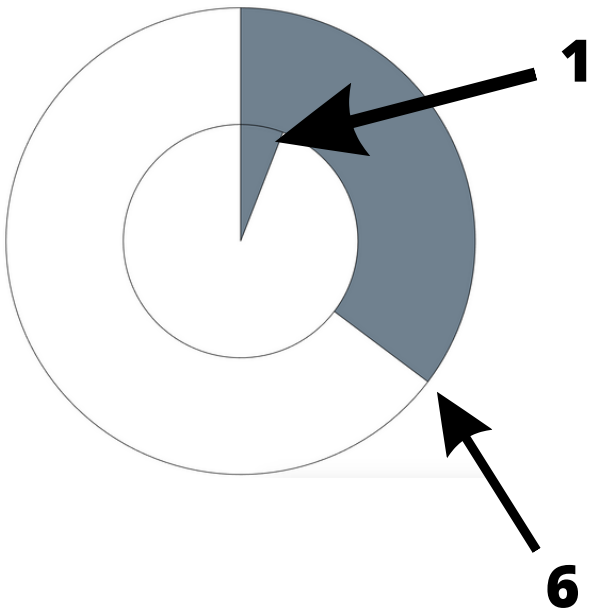
L'hexadécimal est un système de numérotation en base 16 qui utilise les chiffres de 0 à 9 et les lettres de A à F pour représenter des nombres ou des caractères. Tout caractère ASCII se code sur 2 caractères hexadécimal (voir table ci-dessous).

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[END OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

Pourquoi l'hexadécimal ? Dans le contexte de mon projet, l'utilisation de l'hexadécimal permet de simplifier la lecture et la conversion des données encodées sur le cercle. Car diviser les 360 degrés d'un cercle pour chaque caractère ASCII aurait créé des parties trop petites, donc difficile à lire. Alors que l'hexadécimal permet de diviser ce cercle en uniquement 17 parties, soit 21 degrés. En effet, chaque partie de 21 degrés du cercle représente une valeur hexadécimale distincte, ce qui facilite grandement la lecture et la conversion de l'information.

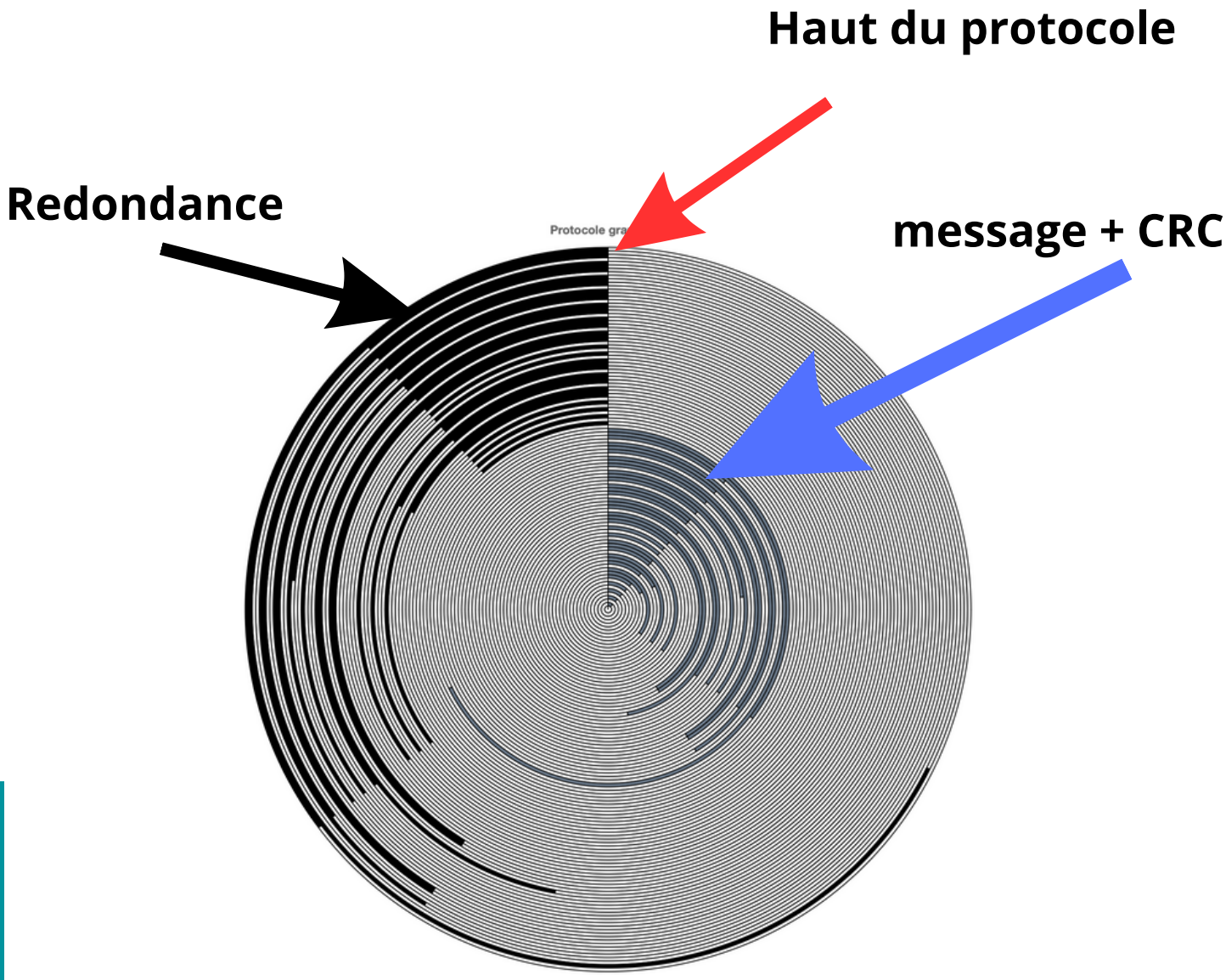
01. Exemple pour le message "a" :



L'héxadécimal de "a" est 61. Donc dans cet exemple généré avec le bouton "encodage simple" on peut voir les 2 arcs de cercle. Celui le plus à l'extérieur correspond à 6 comme on peut le voir dans la figure au dessus. Et celui plus au centre correspond à 1. On retrouve bien l'hexa 61 !

ENCODAGE

Ainsi une fois le message encodé, ce sera une composition de couple d'arcs de cercle chacun correspondant à un caractère ASCII. Ces arcs de cercle ne sont pas dans l'ordre d'apparition du message, et sont doublé, nous verrons pourquoi dans la partie "détection d'erreurs".
Pour trouver le sens correct de lecture il suffit de regarder où commence les arcs de cercle. Dans l'exemple ci-dessous on peut voir clairement le haut indiqué par la flèche rouge.



Dans l'exemple au dessus, nous pouvons voir un encodage complet avec detection d'erreur. La flèche bleue indique la partie grise qui correspond à l'hexadécimale du message + le CRC (que nous verrons plus tard). Et la flèche noir indique le redondance inversé que nous verrons également plus tard.

UTILISATION

01

Dans un premier temps :

- Ouvrez le fichier html dans le navigateur de votre choix.

02

Choisissez votre type d'encodage :

- Bouton "encoder", pour encoder le message avec toutes les fonctionnalités.
- Bouton "encodage simple", pour encoder le message sans détection d'erreur. Sert uniquement à aider à comprendre le fonctionnement

03

Cliquez et découvrez l'encodage !

- Pour encoder une seconde fois, faire F5 ou (cmd + R sur mac)
- Si cela ne marche pas faire CTRL + F5

INFORMATIONS SPECIFIQUE AU PROTOCOLE

Pour garantir un graphique cohérent et lisible, j'ai choisi de limiter la longueur du message à 26 caractères. Lorsque le message est plus court que 26 caractères, les champs vides sont remplis avec le caractère hexadécimal "20" (espace) pour maintenir la taille du message. Lors du décodage, tous les espaces en fin de message sont supprimés et le message initial est récupéré correctement.

La taille du message serait facile à récupérer à l'aide d'un algorithme qui connait les positions et leurs correspondances et avec un calcul simple. Il aurait juste besoin de récupérer les caractères soit de la partie noire, soit de la partie grise puis de les remettre dans l'odre. Supprimer tout code "20" non suivi d'un autre code. Puis de diviser par 2 (chaque ASCII = 2 caractère hexadécimal) et de soustraire 4 (CRC).

Dans chaque cellule (arc de cercle) est codé un 1 caractères hexa soit 4 bits. Deux cellules sont nécessaires pour codé un caractère ACII.

DÉTECTION D'ERREURS

La partie détection d'erreur est très importante dans un protocole graphique. Elle grandement responsable de la complexité de tout protocole de communication. En effet les systèmes de communication les plus aboutis de nos jours ont une détection et correction d'erreur très puissante.

ORDRE D'APPARITION

L'ordre d'apparition des caractères encodés peut paraitre anodin néanmoins on peut déjà s'en servir pour éviter la perte totale du message.

Petit exemple pour mieux comprendre :

Pour simplifier l'exemple, nous ne prenons pas en compte le codage et décodage du message.

Message : **"Salut ça va ?"**

Nous supposons qu'à la réception il y a une perte des 4 premiers caractères. Cette perte peut être dû par exemple à une obstruction d'une partie du protocole (un objet cache l'endroit où sont situés ces informations).
Dans le cas où l'on envoie le message dans l'ordre :

Message envoyé :

Salut ça va ?

Message reçu :

t ça va ?

Ici nous pouvons voir que la perte des 4 premiers caractères a un énorme impact car il devient très difficile de retrouver le premier mot avec seulement une lettre restante. Sachant qu'à la réception on ne sait pas si il y a eu une perte et si il y en a une où est elle située dans le message.

Maintenant, au lieu d'envoyé le message tel quel, nous allons modifier l'ordre d'apparition des caractères. Pour cela nous mélangeons tout simplement les caractère. La logique suivi pour changer l'ordre est bien sûr connu par le receveur du message pour qu'il saches comment remettre les caractères dans l'ordre.

Dans le cas où l'on envoie le message dans l'ordre :

Message envoyé :

auraS çt lv ?

Message reçu

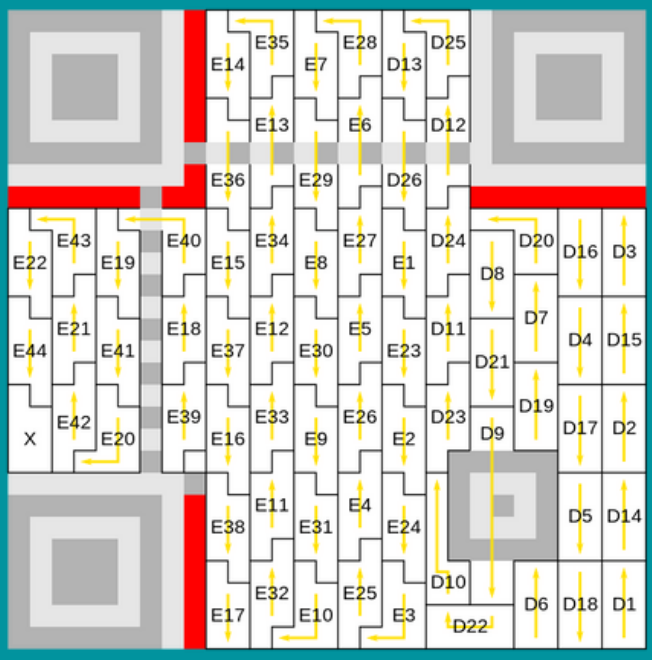
après remise

dans l'ordre:

Slt ç v ?

Grâce à cette méthode, la perte des 4 premiers caractères n'agit pas sur un seul mot mais réparti la perte sur différent mots. Ainsi nous ne perdons pas un mot complet et il sera plus facile de retrouver le message initial. Cela ne garantit pas la correction d'erreur mais cela contribue.

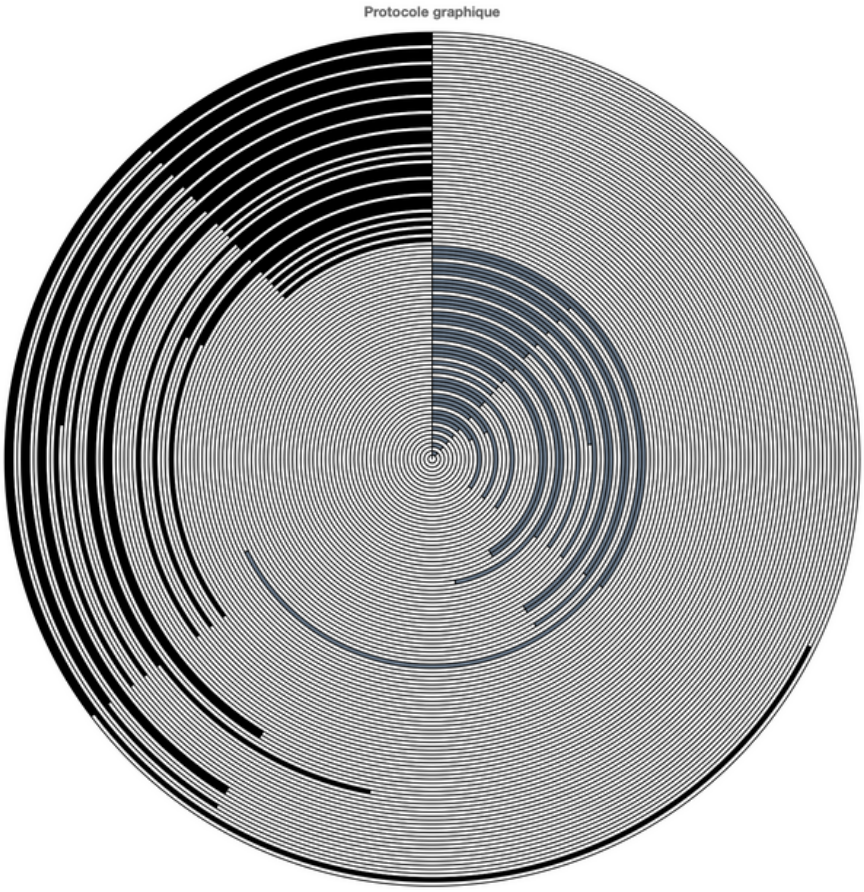
J'ai donc ajouté cette méthode à mon protocole en me basant sur l'ordre d'apparition du QR code.



Sur ce schéma, les cellules D contiennent les données. Et on peut donc voir que en bas à gauche l'ordre d'apparition n'est pas 1, 2, 3, 4 etc mais que les cellules sont mélangés d'une certaine façon. C'est donc cet ordre là que j'ai suivi pour mon protocole.

REDONDANCE

Ensuite, j'ai également inséré de la redondance dans le protocole. C'est à dire que le message est présent plusieurs fois. Ainsi en cas de perte de partie du message, il est possible de faire une correction d'erreur en se basant sur les autres apparitions du message. Cette redondance est une copie inversé du codage.C'est à dire que le cercle part dans le sens inverse des aiguilles d'une montre. Ce qui permet d'opposer la position des mêmes informations dans le protocole. Ainsi la perte de la même information est peu probable.



Dans la figure ci-dessus, la partie grise est le codage du message, et en noire la redondance inversé. On peut voir que les informations ne sont pas situés au même endroit donc en cas d'obstruction du protocole, il est peu probable que cela cache l'information dans la partie grise et la partie noire. Ainsi la perte de l'information est grandement réduite.

CYCLIC REDUNDANCY CHECK (CRC)

Au début de ce projet, j'ai envisagé d'implémenter le code de Hamming pour ajouter une redondance avec correction d'erreur à mes transmissions de messages en hexadécimal. Cependant, j'ai rapidement réalisé que cette méthode n'était pas compatible avec mon système de transmission qui utilise uniquement l'hexadécimal. J'ai donc effectué des recherches pour trouver une autre méthode de détection et de correction d'erreur compatible avec l'hexadécimal. C'est ainsi que j'ai découvert la méthode de CRC, qui permet d'ajouter un checksum au message en hexadécimal et de vérifier son intégrité lors de la réception. J'ai alors décidé d'implémenter cette méthode dans mon système de transmission pour assurer une transmission fiable de mes messages.

Le CRC (Cyclic Redundancy Check) est une méthode de détection d'erreur utilisée pour détecter les erreurs de transmission de données. Pour ajouter le CRC à un message en hexadécimal, la première étape consiste à convertir le message en une suite de bits. Ensuite, on ajoute un nombre de bits supplémentaires, appelés bits de redondance, à la fin du message, de manière à obtenir un message plus long. Le nombre de bits de redondance ajoutés dépend du degré du CRC, qui est déterminé à l'avance.
Ensuite, on effectue une division binaire du message plus long par un polynôme générateur pré-défini. Le résultat de cette division est appelé le reste, qui est un nombre de bits équivalent au degré du polynôme générateur moins un. Ce reste est ajouté à la fin du message initial pour former le message final, qui inclut le CRC.

De même, lors de la réception du message hexadécimal, le CRC est calculé à nouveau sur la séquence de bits correspondante, et comparé au CRC reçu avec le message. Si les deux CRC sont identiques, le message est considéré comme valide, sinon il est considéré comme corrompu. La fonction de vérification du CRC a été ajouté au code mais n'est pas utilisé car décodage non demandé.

Le CRC et le code de Hamming sont tous deux des méthodes de détection et de correction d'erreurs dans les données transmises. Cependant, le CRC est plus adapté pour les messages de taille importante, tandis que le code de Hamming est plus efficace pour les messages de petite taille. De plus, le CRC nécessite l'utilisation d'un polynôme générateur spécifique pour chaque application, tandis que le code de Hamming a un algorithme universel pour tous les messages de taille fixe.

DÉVELOPPEMENT

LANGAGE UTILISÉ

Pour le développement, j'ai décidé d'utiliser le langage JavaScript avec Chart.js afin d'afficher mon protocole en HTML. Chart.JS est un framework qui permet de créer des graphiques interactifs et personnalisables pour le web. Elle offre une grande variété de graphiques tels que les diagrammes en barres, les camemberts et les nuages de points. J'ai utilisé cette bibliothèque pour créer mes cercles en utilisant le graphique de type "Multi Séries Pie" proposé par le Framework.

ORGANISATION DU PROJET ET DU CODE

Mon projet est composé uniquement de 2 fichiers :

01. Fichier HTML

Ce fichier a pour seul but de permettre de rentrer le message, d'appeler le script JavaScript et d'afficher une fois encodé.

02. Fichier JS

Ce fichier contient tout le code utilisé pour encoder le message. Il est réparti en différentes étapes, fonctions. Tout le code a été commenté, afin d'expliquer le rôle de chaque fonction.

FONCTIONS PRINCIPALES

Présentation des fonctions principales du projet :

01. Encoder

La fonction encoder est comparable à une fonction main. C'est elle qui est lancée lorsqu'on clique sur les boutons pour encoder. Elle ve recevoir le message brut est appelé chaque étape d'encodage pour enfin lancer l'affichage.

02. AsciitoHexa ou stringtoHex

Ces 2 fonctionsconvertissent en hexadécimal. L'une converti un seul caractère ASCII alors que l'autre converti tout une chaine de caractère.

03. dataBuilder

DataBuilder est une fonction qui va prendre en entrée un code hexa et qui va calculer et renvoyer ces données correspondantes en pourcentages, pour réglé l'avancement de l'arc de cercle.

04. addDataset

Un dataset est le format de donnée qui contient tous les paramètres pour afficher correctement nos données avec le Framework Char.JS.Ici addDataset va créé le dataset correspondant et l'ajouter à la liste des datasets.

05. afficher

Afficher est la fonction qui va créé le graphique avec toutes les données compris dans la liste des datasets.

AVANTAGES ET INCONVÉNIENTS

AVANTAGES

- Ce protocole se démarque par son originalité, car il ne se limite pas à une simple matrice carrée, comme c'est souvent le cas. J'ai souhaité innover en utilisant une forme circulaire ainsi qu'en remplaçant le binaire par l'hexadécimal.
- Grâce à l'utilisation d'un CRC, d'une redondance et d'un mélange d'ordre d'apparition des données basé sur le QR code, la détection d'erreur est performante et complète.
- Sa forme circulaire le rend facilement identifiable dans une page remplie d'informations.
- Les couleurs choisies, le gris et le noir, sont aisément distinguables et ne seront pas affectées par des variations d'intensité lumineuse ou d'autres perturbations.

INCONVÉNIENTS

- La lisibilité peut être compromise par la longueur du message, car chaque arc de cercle occupe une quantité importante d'espace en plus de la redondance. Donc dans le cas d'un message long alors chaque cellule deviennent plus complexe à décerner.
- De plus, la longueur des arcs de cercle peut aussi être difficile à lire.
- Enfin, impossible d'y ajouter un logo. En effet le bonus d'ajouter le logo n'a pas été possible car pas d'espace pour intégrer une image. En remplacement j'ai ajouté une double fonctionnalité d'encodage qui permet d'afficher un message "simple".

THÉO NICOLAS

GROUPE : G3

2022/2023

Coordonnées :

theo.NICOLAS@etu.uca.fr