Rapport de TP de Réseaux II

Emilie Péret Loris Croce

Fonctionnement du protocole

Nous avons opté pour un protocole faisant usage de 5 couleurs, ici en hexadécimal :

- `#66CCE1`
- `#AA7DFC`
- `#B2F64C`
- `#FF5794`
- `#FAA23D`

Les caractères du message sont représentés sous forme de "carrés" de 5×5 pixels. Où les 0 et les 1 sont sont représentés par des couples de couleurs, celles-ci changent pour 1 caractère sur 2, ce qui permet un décodage aisé en séparant les caractères. La dernière couleur reste disponible pour un éventuel logo.

Le tout premier "carré" est une *mire* représentant l'ensemble des couleurs utilisées (une par ligne). Il permet de donner l'orientation de lecture du code car il est placé en haut à gauche ainsi que la taille d'un pixel de par les lignes dessinées.

Les caractères sont codés sur $16bits$ cela permettant une bonne prise en charge des caractères spéciaux. Ils sont ensuite transformés par code de *hamming* ce qui ramène la taille d'un caractère à $21bits$. Sur un carré de 5×5 on a donc 4 "pixels" libres pour un logo.

Technologie utilisée

Pour ce TP, nous avons choisi d'utiliser le langage JavaScript pour plusieurs raisons. D'une part son haut niveau permettait de s'abstraire de certaines contraintes telles que le typage ou la gestion de la mémoire. Mais c'est aussi pour ses propriétés inhérentes, en effet, fournir une interface sous la forme d'une page HTML simple nous a paru une solution efficace. De plus, JavaScript permet de dessiner dans un *canvas* déclaré dans la page.

Toutefois si JavaScript a pour défaut de ne pas être aussi performant (car haut niveau) que certains autres langages (comme le C par exemple) cet inconvénient était négligeable ici étant donné la faible complexité des opérations tout en étant dans une situation où l'instantanéité du programme n'est pas à maximiser.

Implémentation

Côté HTML Nous avons déclaré simplement une un groupe d'éléments pour les entrées alignés à gauche composé d'une zone de saisie de texte, et de deux boutons : *Valider* et *Supprimer*. À droite un *canvas* comme précisé ci-dessus destiné à afficher le code graphique.

Traitement du message Les événements du et éléments du DOM[1] ont été récupérés respectivement par les méthodes `addEventLister(`*`event, function`*`)` et `document.getElementById(`*`id`*`)`. Ceux-ci ont étés stockés et assignés à des variables. Ensuite des traitements sur le message ont étés effectués pour parvenir à une forme interprétable pour notre protocole. Pour cela un ensemble de fonctions ont été créées :

- `getBytes` permettant de récupérer les codes unicode d'un chaîne.
- `toBits` permettant de transformer ces codes sous forme binaire.
- `fixBits` normalisant les nombres binaires en rajoutant si nécessaire des 0 supplémentaires par rapport on nombre fixé en paramètre.
- `hamming` effectue une correction de hamming sur un nombre binaire.
- `toMatrix` transforme le tout en une matrice pour être dessinée.

1. Document Object Model

Affichage Pour gérer l'affichage du code de fonctions ont été créés : `draw` et `clear` permettant respectivement d'afficher et supprimer un code correspondant à au message entré. Il faut au préalable assigner le contexte du canvas à une variable, ici `ctx`. Pour dessiner –ici les "pixels"– on dispose d'une fonction fournie par le canvas : `ctx.fillRect(`*`x,y,width,height`*`)` permettant de dessiner un rectangle –ici carré– et `ctx.fillStyle` permettant auquel il est possible d'assigner la couleur voulue. Pour dessiner le code on dessine d'abord la mire "en dur", puis on parcoure le tableau `bits` en 3 dimensions pour créer les carrés bits par bits. On sélectionne la couleur en vérifiant le contenu de la cellule et grâce à une variable booléenne permettant le changement de couples de couleurs entre chaque caractère.

Améliorations, remarques

En premier lieu, il est important de rappeler que le code devait avoir à priori une forme s'approchant d'un carré (ou au moins s'afficher sur plusieurs colonnes), cependant cela n'a pas été mis en place car les tentatives n'ont pas permis de former de revenir à la ligne au moment opportun en fonction de la taille du message.

Ensuite, afin d'améliorer le protocole la taille des pixels aurait pu être inversement proportionnelle à la taille du message. Une correction d'erreur variable ainsi qu'un choix des couleurs aurait également pu être implémenté. Pour finir, les pixels inutilisés aurait pu être déplacés dans chaque caractère codé pour retranscrire un logo.