Premier rapport de projet Arduino

EN COURS :

Nous avons travaillé ensemble afin de s'approprier le fonctionnement de notre écran. Plus concrètement, le branchement ne nous a pas posé de difficultés, mais réussir à afficher ce que l'on souhaitait était déjà plus ardu.

Nous avons commencé par exécuter le code d'exemple donné avec la bibliothèque (ou librairie) que nous utilisons, à savoir le graphicexemple.ino de ILI9341_t3. Tout marchait à merveille.

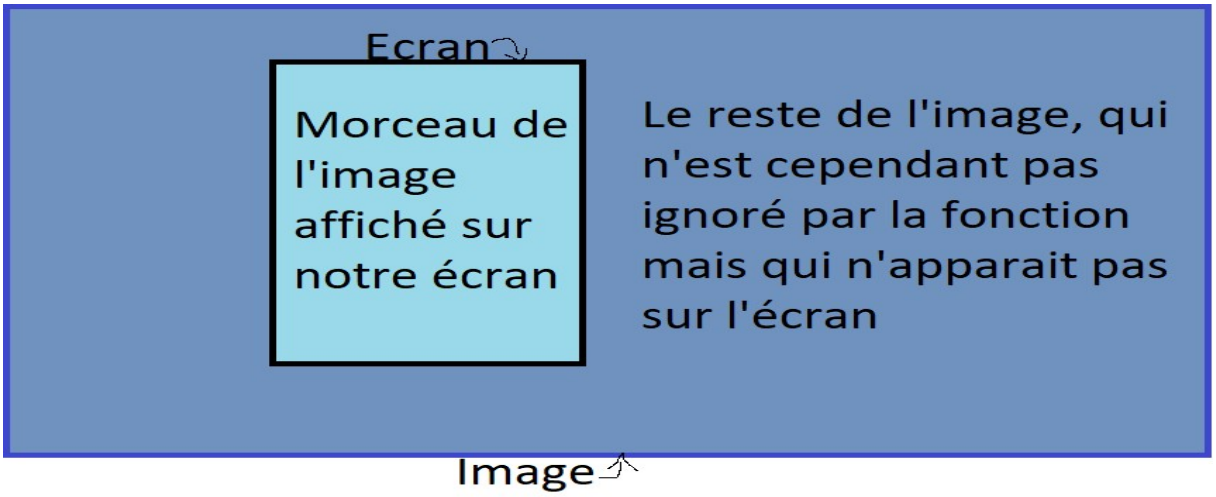
Nous avons par la suite cherché à dessiner sur l'écran les images de nos rêves à partir de fichier au format bitmap (.bmp) aux couleurs encodées sur 24 bits. Encore une fois, miracle, tout fonctionne (alors que j'avais eu bien du mal à faire ne serait-ce qu'allumer l'écran des mois plus tôt).

Un problème se posait alors : la vitesse d'affichage. Notre écran a une résolution de 240x320 et oui, cela fait beaucoup de pixels. De ce fait, afficher une image de cette même résolution sur notre écran prenait pas moins de 500ms ! Difficile de se satisfaire de cela quand l'on souhaite aboutir à un jeu vidéo : 2 images par secondes (ou fps) n'est pas tolérable.

Nous nous sommes alors intéressé à d'autres moyens d'afficher des images complexes sur l'écran, il n'était pas question de les dessiner pixel par pixel dans un code dont des centaines de lignes seraient consacrées aux images seulement. Nous avons trouvé un autre moyen de stockage d'image nommé « PROGMEM », qui consiste en une sorte de « tableau de couleurs » contenant chacune des couleurs de l'image que l'on souhaite dessiner, encodées en hexadécimal et stocké dans la mémoire flash au lieu de la ram. Il semblerait que ce soit plus rapide, mais pour vérifier il nous faut d'abord convertir notre image en cette sorte de tableau, un autre problème. Internet a rempli son rôle et nous a redirigé vers un petit logiciel qui fait, entre autre, cela très bien : lcd-image-converter.exe
https://sourceforge.net/projects/lcd-image-converter/files/latest/download

Nous avions fait des tests, et il semblerait en effet que ce soit plus rapide, pour la même image nous sommes maintenant passés à 300ms. C'est mieux, toujours trop lent, mais mieux. Ce procédé a cependant un avantage : la fonction permettant de dessiner une image peut en dessiner une plus grande que l'écran, ce qui aura pour effet de dessiner un fragment de l'image de la taille de l'écran, ce qui sera utile pour les déplacements.

Illustration à la page suivante.



Nous n'étions pas sortis d'affaire pour autant, car en plus de la durée toujours trop longue, un autre problème se pose : les images étant contenues dans la mémoire flash du microcontrôleur utilisé (à savoir, la Teensy 4.0), le poids du code devenait conséquent avec presque 65% de la place occupée par une image de 3000x3000 pixels. Nous corrigerons ce problème plus tard avec de la lecture SD, nous avons préféré nous concentrer sur la rapidité d'affichage d'une image sur l'écran.

Le cours était sur le point de se terminer, notre idée était de n'afficher la carte du jeu (car oui, c'est elle l'image dont je parle depuis le début du rapport) sur une partie de l'écran réduite à 240x240 pour l'afficher plus rapidement, et profiter de l'espace gagné pour en faire un menu ou quelque chose d'utile tout de même.

HORS COURS :

J'ai tout simplement débuté le code du jeu brut, à savoir la définition des classes. Je compte terminer cela au plus vite pour la semaine prochaine afin de remplir ma part de travail sur le diagramme de Gantt. Je ferai peut être également une documentation pour mieux comprendre le code, ce qui me sera utile et sera également utile à Matéo.

Ce début de code n'étant pas encore très prometteur, concentrons-nous sur la seconde partie de mon travail : optimiser la vitesse d'affichage d'image sur l'écran. J'ai pensé à optimiser la fonction utilisée dans l'exemple de la bibliothèque qui nous permet de dessiner sur l'écran avec la Teensy … puis je me suis exécuté.

J'ai passé deux heures dessus, et je suis finalement abouti à une fonction améliorée qui dessine une image (malheureusement, à des coordonnées fixes pour le moment : pas de déplacement possible) en un temps records : 50ms seulement au lieu des 300 précédentes ! Un plaisir de voir les images se dessiner si vite !

Matéo m'a rejoint à ce moment-ci, et nous avions amélioré puis terminé cette fonction qui permet maintenant de dessiner une image de dimensions carrée (autant de pixels en largeur qu'en hauteur) à des coordonnées différentes, et cela en 60 ms. Ces 10 ms supplémentaires ne sont pas perdues, elles permettent de dessiner le personnage une dizaine de fois pendant le dessin de la carte afin de ne pas avoir une sensation de « clignotement » sur le personnage. En effet, si nous redessinons le

personnage sur la carte à des intervalles de temps trop grandes, l'œil s'en aperçoit er nous fait croire à un clignotement. C'est un problème réglé contre 10 petites millisecondes.

Par ailleurs, le dessin du personnage est détouré. C'est-à-dire qu'il y a un effet de transparence, il n'est pas dessiné comme un carré sur l'écran avec des pixels noirs ou blancs pour remplir les cases non remplies par son dessin, mais il est dessiné comme tel sans endommager le magnifique décor.

Les pixels noirs ne sont pas dessinés sur l'écran : nous y voyons la carte au travers pour donner un effet de superposition du personnage sur la carte.

Finalement, nous avons passé le reste de la soirée à essayer d'élargir les possibilités de la fonction à des images de dimension non carrées, jusqu'à ce que nous nous sommes rendus compte que c'est strictement impossible, puisque le type utilisé en PROGMEM « formate » sa liste de couleurs hexadécimales en une listes de dimension carrée. Précisons que le dessin est évidemment de Matéo.

BIBLIOGRAPHIE :

Nous avons préparé plusieurs documents sources qui viennent appuyer nos rapports. Vous pouvez dès à présent jeter un coup d'œil à différentes photos et vidéos appuyant nos hauts faits sur la branche « feat » de notre GitHub, dans le dossier « documents supplémentaires ».