Deuxième rapport de projet Arduino

EN COURS :

Nous avons utilisés ces 3 heures à notre disposition afin de mettre au point la chatbox du jeu, c'est-à-dire la bulle de dialogue dans laquelle les textes que disent les personnages défilent. C'était bien plus compliqué que prévu, et ce notamment à cause d'un problème qui nous cause du tort depuis une semaine : notre Teensy n'en fait qu'à sa tête et décide parfois de ne plus être reconnus comme connecté à un quelconque port USB. Problématique, n'est-ce pas ?

C'est pourquoi nous avons dû coder différentes fonctionnalités individuellement, dans des codes différents, afin de ne pas surcharger la Teensy, qui nous résiste. Nous parlerons plus en profondeur de ce problème dans la section « Hors cours ».

Nous avons donc ouvert un nouveau croquis, configuré l'écran et passé un peu moins de deux heures à mettre au point une chatbox qui répondait à nos attentes. Le texte s'y défile avec fluidité, ce que la bibliothèque ILI9341_t3 n'avait pas prévu, ou presque.

En effet, sans pour autant trop rentrer dans les détails, le fonctionnement de l'écriture et du dessin sur l'écran fonctionne par un système de coordonnées couplé à une notion de « pointeur » : pour dessiner des figures primitives géométriques telles que des rectangles, des cercles ou des carrés, des coordonnées suffisent. En revanche, pour certaines raisons qui ne me semblent pas logique du tout, lorsqu'il s'agit d'écrire du texte sur ledit écran, le fonctionnement est différent. Il faut en effet indiquer vers quelles coordonnées (x,y) pointer avec l'appel d'une méthode, puis écrire sur l'écran le texte que l'on souhaite. La partie bien faite de cette bibliothèque, c'est le retour à la ligne automatique. Il faut l'admettre. Cependant, lorsque le texte devient conséquent, l'écriture se ralentit au fur et à mesure que la chaine de caractère défile sur l'écran, et c'est le point noir de trop. Nous avons donc décidé de réécrire une fonction permettant d'écrire du texte à l'écran, qui permet d'écrire un texte petit à petit sans décroître en vitesse. Pour cela, nous avons simplement déplace le pointer de texte dynamiquement en fonction du nombre de caractère entré, pour n'avoir qu'à écrire les nouvelles lettres toutes les 250 boucles sans réécrire l'ancien texte (ce qui peut paraitre absurde, mais c'est le fonctionnement de la fonction de la bibliothèque utilisés).

Je suis peut-être entré dans les détails finalement, mais l'idée est là : nous avons modifié une nouvelle fois la bibliothèque permettant de dessiner sur l'écran.

Nous sommes assez fiers du rendu, mais nous le sommes plus encore pour ce qui a été accomplis hors cours depuis la semaine dernière. Se rassembler plusieurs fois nous a permis d'avancer considérablement dans un domaine que nous ne maitrisons que trop peu.

HORS COURS :

J'ai dû relire le dernier rapport fait pour savoir ce qui est nouveau depuis lors.

Nous avions réécris, la semaine dernière, la fonction permettant de dessiner une image depuis la mémoire de la Teensy pour accélérer près de 8 fois le processus. Cette fois-ci, j'ai réécris celle permettant de dessiner une image depuis la carte SD. Elle n'a pas gagné en vitesse, mais elle permet quelque chose d'assez inédit : dessiner une image détourée, ce qui sera très utile pour dessiner les différents Arduimon lors des combats.

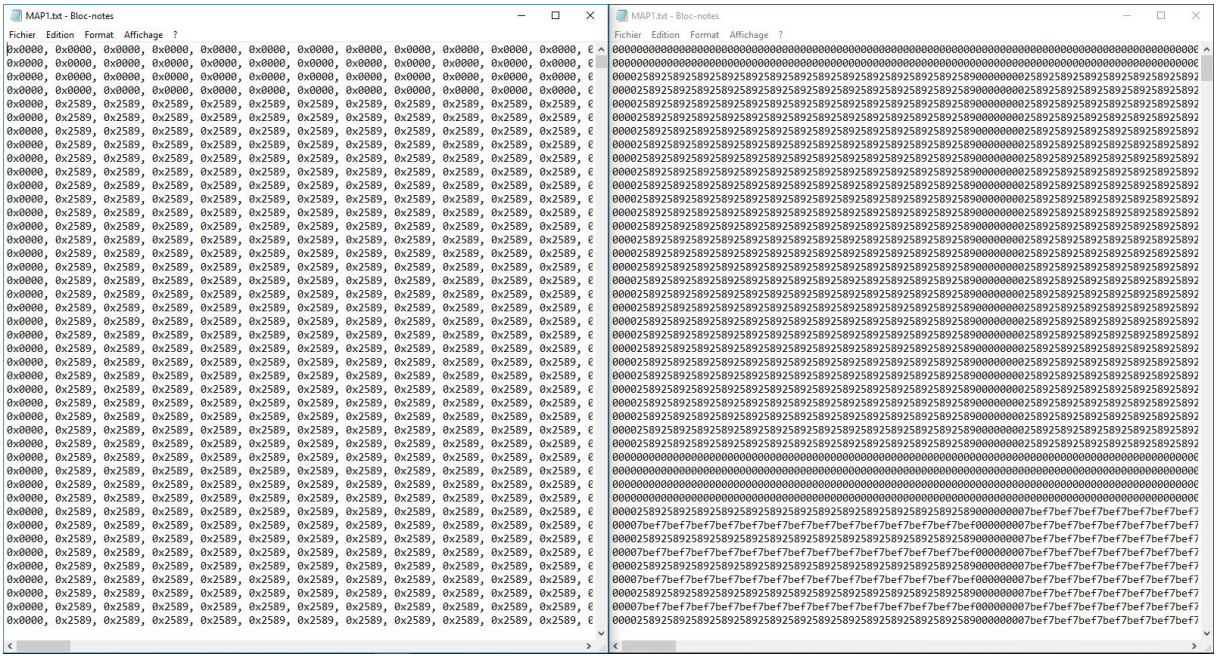
Idem pour les images stockées dans la mémoire de la Teensy, j'ai réécris une fonction qui permet de les dessiner détourées. Cette dernière est utilisée pour dessiner notre personnage sans qu'il y ait un énorme carrée blanc ou noir autour de lui.

Ces réécritures s'accompagnent de changement importants dans la bibliothèque utilisée, notamment vis-à-vis de la méthode de dessin : au lieu de dessiner une image point par point (pixel par pixel), nous le faisons désormais lignes par lignes, d'où l'amélioration de la vitesse.

Nous avons également mis au point un changement de la carte de jeu, et ce par une fonction, grâce à la communication SD. Nous stockons dans la carte SD les informations relatives aux couleurs de pixels d'un tableau de jeu, et nous remplaçons la carte actuelle par la nouvelle. Très pratique, et aussi plutôt rapide : 2.5 secondes en moyennes pour une carte de 480x480 pixels.

J'ai aussi développé un petit outil en python permettant de diviser par deux la taille d'un fichier contenant les caractères utilisés pour la lecture de carte SD décrite plus tôt. Le fonctionnement est assez simple :

Les chaines de caractères représentant nos images sont au format 16-bit colored, c'est-à-dire de la forme : « 0x0000, 0x7825, 0xf7892, … ». J'ai remarqué la récurrence des 4 caractères «, 0x» et l'importance des 4 autres qui varient (ceux suivant le « 0x »). En réécrivant un autre fichier sans les virgules, espaces et « 0x », nous passons donc de 8 caractères par pixels à 4. Ci-joint une capture d'écran illustrant mes propos :



A gauche se trouve le fichier texte que nous renvoie le logiciel que nous utilisons pour convertir des images en chaînes de caractères *lcd-image-converter* (cf : le cahier des charges). Vous pouvez voir à droite ce que produit le script python : un fichier plus condensé, qui contient autant d'information mais plus aucune virgule, espace ou « 0x ». Il nous fallait réécrire nos fonctions de lecture SD, mais les résultats sont très satisfaisant : nous sommes entre autre passés de 2,5 secondes à 1,25 secondes pour le changement de carte (comme quoi, diviser par deux la taille du fichier divise également par deux le temps de dessin. La logique m'étonnera toujours).

Nous avons également mis en place le déplacement du personnage cases par cases (une case, ou « tile », fait 16x16 pixels) via des boutons.

Nous avons très vite animé nos personnages en changeant son Sprite (son image) selon son déplacement.

Enfin, nous avons mis au point des « transitions » de 2px lors du déplacement, ce qui nous donne le calcul suivant :

Une image de l'écran se redessine entièrement en 65 secondes environ, et le personnage se déplace d'une tile à l'autre de 2px par 2 px. Une tile est composée de 16x16px, il nous fait environ 65*16/2 = 520 ms pour avancer d'une case. Cette vitesse nous convient parfaitement.

Notre dernière avancée de la semaine fut l'implémentation de collision. Nous avons pour cela associé à une carte une matrice (tableau de dimension 2) associé aux coordonnées en tiles du jeu. Un 1 signifie qu'il y a une collision, et avant de se déplacer, nous vérifions vers la case vers laquelle on veut avancer nous oppose une collision ou non. Simple comme bonjour.

Parlons légèrement du problème rencontré. Pour certaines raisons, quand le code écris devient trop conséquent, la carte Teensy n'est plus reconnue comme branchée à un port, et plus rien ne fonctionne. Nous avons fait quelques tests avec M.Masson, et nous pensons à une défaillance. La seconde carte est arrivée aujourd'hui, il nous faudra faire quelques tests pour déterminer si le problème venait bel et bien de la carte, cas dans lequel nous saurons quoi faire pour passer outre.

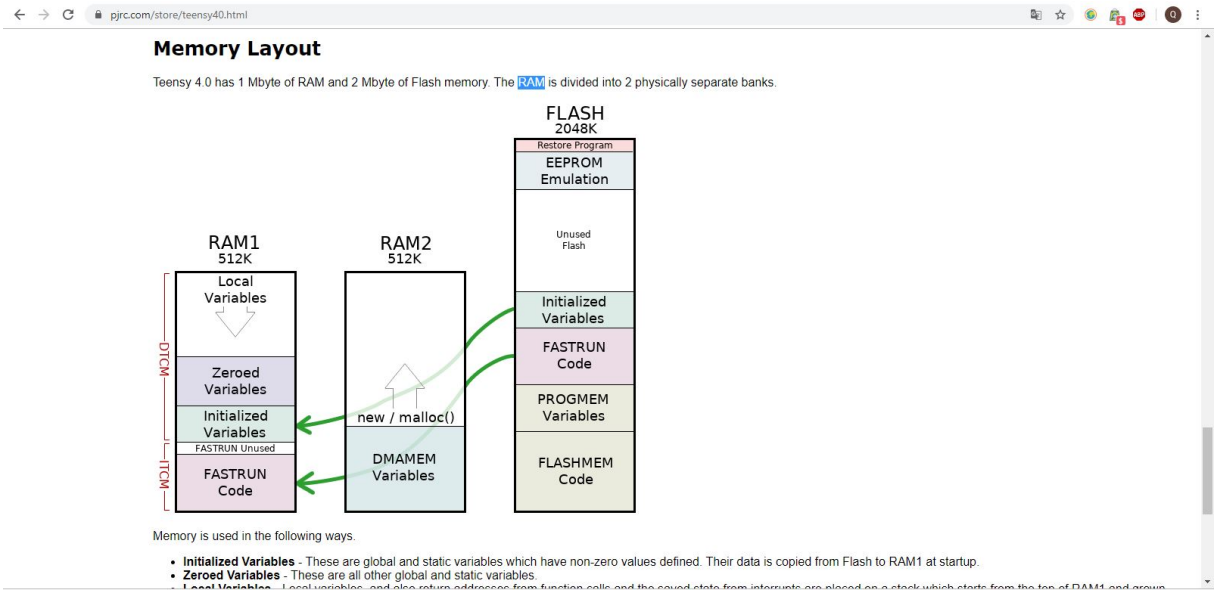
EDIT : Il est 17h31 et nous sommes le 17/12/2019. Avec Matéo, nous avons essayé la nouvelle Teensy, et il semblerait qu'elle aussi fait face au même problème. La première n'était donc pas endommagée.

Cependant, une idée très étrange m'est venue : et si le problème venait de la ram, du stockage de la carte ? En effet, celle-ci nous affichait 49% utilisée, mais ne sait-on jamais. Nous avons réduit d'un facteur 2 la taille de notre carte, et devinez quoi : tout fonctionne. Je me suis alors souvenu avoir lu quelque chose au sujet du stockage non FLASH de la Teensy, et nous avons alors compris.

Dans les faits, nous avons bel et bien utilisé 49% de la mémoire dont nous disposons, mais il se trouve que cette mémoire est divisée en deux parties bien spécifiques à différents types de stockage. Nous avions simplement saturé celle qui se chargeait de mémoriser les variables locales, la DTCM !

De ce fait, faire appel à certaines fonctions ou non permettait de les prendre en compte ou non dans la compilation du code, d'où les problèmes rencontrée si l'on utilisait ou non certaines méthodes ! Cela explique également pourquoi nous pouvions dessiner des images complexes jusqu'alors, mais dessiner un rectangle de plus faisait appel à trop de RAM pour le peu qu'il nous restait. Il semblerait que nous ne nous sommes pas assez renseigné sur le contenu de la Teensy (par « nous » j'entends majoritairement « moi » qui pensait m'y connaitre assez : j'admets avoir été pris d'avance par certaines spécifiés).

Voici une capture d'écran du descriptif de la RAM du modèle 4.0 de la Teensy.



Finalement, nous avions deux solutions potentielles qui s'offrent à nous : Posséder une carte dont les pixels sont codés sur 8 bits au lieu de 16 bits comme nous le faisons actuellement, ou réduire la taille de notre carte.

La première solution s'est avérée impossible, car tous les types uint8, uint16, uint24 et uint32 sont en réalité codé sur 4 octets (la précision du nombre de bit maximal sert simplement à favoriser le portage d'un code. http://en.wikipedia.org/wiki/Stdint.h devrait donner plus d'information à ce sujet si cela intéresse quelqu'un).

Nous devrons donc réduire la taille de notre carte, et par extension de toutes les cartes que nous importerons depuis la carte SD. Disons que la taille maximale stockable localement des différentes cartes de jeu sera amenés à être réduis. Ce n'est rien, nous préférons cela à être bloqué pour si peu. Notre première idée est de passer de 480x480 à 360x360, afin de prendre une bonne marge de RAM pour les futures implémentations. Qui sait, peut-être aurons-nous besoin d'espace à l'avenir.

Puisque nous avons fait beaucoup d'avancées en une semaine, beaucoup d'illustrations de mes dires doivent suivre, et sont décrites dans la partie bibliographie. (Bien que nous devions reprendre notre ancien code avec pleins de problèmes à cause de la mémoire pour vous fournir un support, car nous n'aurons pas le temps de tout corriger avant la deadline de rendu de rapport). Je vous suggère de jeter un œil au « Rapport 2 ».

PS : l'ironie du sort a bien entendu fait en sorte que nous ayons des problèmes de lecture avec la carte SD. Le changement de carte et le dessin d'Arduimon depuis une carte SD devra donc attendre la semaine prochaine. Nous subissons le Karma.

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 ».