Récupérez les sprites des logiciels en HIRES

par André C.

En mode HIRES, pour afficher quelque chose qui bouge rapidement (cas des jeux d'arcade), il faut obligatoirement passer au langage machine.

La commande Basic CHAR est trop lente pour faire évoluer des sprites basés sur des caractères redéfinis. Or les astuces développées pour stocker, afficher et déplacer des objets en mode HIRES restent mystérieuses tant qu'on ne procède pas à un "reverse engineering". Et c'est un gros boulot !

Oui, mais il n'est pas sûr que l'on soit obligé d'en arriver là. Notre propos n'est pas de comprendre comment marche le programme, mais seulement de récupérer le dessin des sprites et les data nécessaires à leur affichage.

Lors de la sortie de Xenon-1, je me souviens avoir été fasciné par la beauté et la rapidité des sprites, lesquels sont animés qui plus est ! Je ne peux pas faire autrement que de choisir cet exemple pour illustrer mon propos sur la récupération des sprites.

Un écran HIRES est composé de 200 lignes de 40 octets. Ces 8000 octets sont, soit des attributs, soit des octets compris entre #40 et #7F. Dans ce dernier cas, chaque octet représente un "tiret" composé de 6 pixels.

Récupérer les data d'un sprite, c'est récupérer les octets correspondant à tous les tirets qui le composent et si le sprite est multicolore, la valeur et la position des attributs utilisés. En outre, si le sprite est animé, il faut récupérer toutes les images du sprites.

Je vous propose deux méthodes :

1. Faire simultanément un dump de la mémoire et une recopie d'écran (enfin, aussi simultanément que possible). Avec un éditeur hexadécimal, comparer les octets de la zone écran HIRES avec les objets présents sur la recopie d'écran. Récupérer tout simplement ces octets, qui constituent les data de définition des sprites.
2. Faire une recopie d'écran. Agrandir le fichier .bmp ou .png jusqu'au niveau du pixel avec un bon logiciel de traitement d'image. Isoler le sprite intéressant et le numériser manuellement. La bonne vieille méthode quoi ! Faisable mais ch…, heu… fastidieux ! Si vous avez beaucoup de sprites à traiter vous pouvez aussi utiliser une feuille de calculs pour simplifier la numérisation.

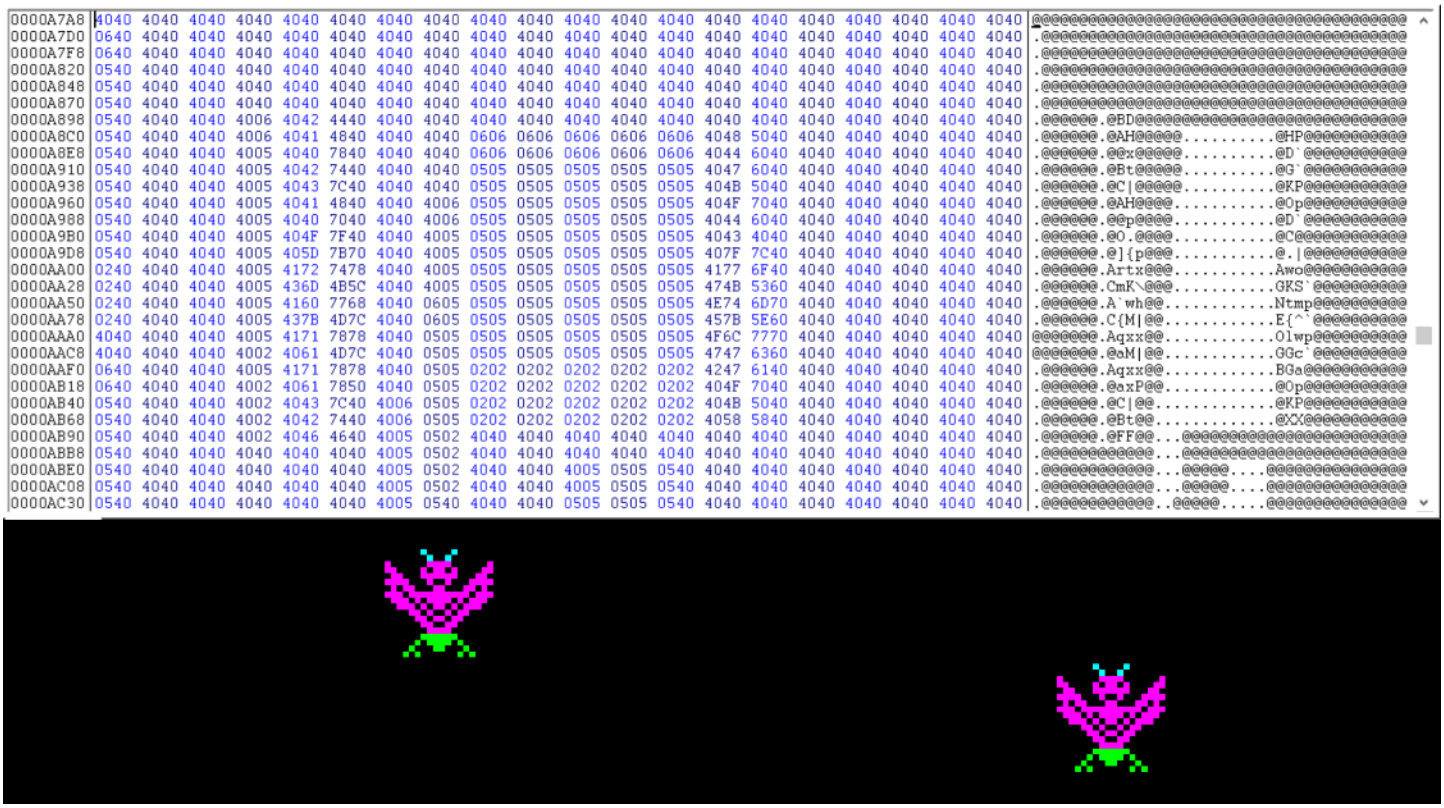
La première méthode en pratique

Les paresseux sont des gens qui cherchent à s'éviter du travail. C'est en cherchant à échapper à la phase de numérisation des tirets que j'ai eu l'idée de la méthode 1, dérivée de celle que j'ai utilisée précédemment pour les écrans TEXT.

Autant vous avouer tout de suite, que ma paresse n'a pas été récompensée ! Certes cette méthode marche, mais au final, c'est très compliqué. Vous pouvez donc passer directement à la seconde méthode, sauf si vous avez la fantaisie et le temps d'être curieux !

Procédure dans Euphoric

1. Lancez Xenon1.tap avec Euphoric en configuration Atmos (vous trouverez la version Simon G. de ce tap dans le zip qui accompagne cet article, ainsi que sur le site oric.org).
2. Allez à l'écran qui vous intéresse, le premier par exemple.
3. Pressez aussi rapidement que possible les touches F9 (dump de la mémoire) puis F12 (recopie de l'écran).
4. Pressez F11 (débogueur) pour figer le jeu et gardez la fenêtre en réserve en vue d'un autre essai (il s'agit de sprites animés et il y a plusieurs images à récupérer pour chacun).
5. Renommez le fichier DUMP qui a été généré par exemple en DumpX1-01.bin
6. Renommez le fichier Screenxx.bmp en Xenon1-01.bmp



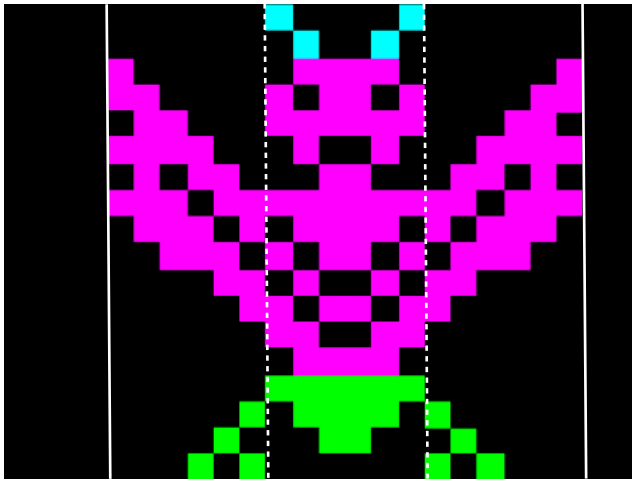
Procédure avec votre éditeur hexadécimal

1. Ouvrez DumpX1-01.bin et réglez l'affichage pour avoir 40 octets en largeur.
2. Allez à l'adresse #A000 (début de l'écran HIRES). Chaque ligne de l'éditeur correspond à une ligne HIRES.
3. Dans la partie gauche de l'éditeur se trouvent les octets présents dans l'écran. Chacun représente, soit un attribut, soit un tiret de 6 pixels, selon que son bit 6 est à zéro (attribut) ou à un (tiret). Dans ce dernier cas, les 6 bits les plus faibles représentent les pixels. Par exemple les octets #40 (les plus nombreux et de loin) soit 01000000 en binaire, correspondent à un espace ou plutôt à un tiret dont les 6 pixels sont de la couleur du fond, donc invisibles (par défaut en HIRES, le fond est noir et l'encre blanche).
4. Dans la partie droite de l'éditeur, sont affichés les caractères Ascii correspondants. Le foisonnement de @ qui correspond aux nombreux octets #40 saute aux yeux. On peut voir aussi soit des points (les attributs), soit des caractères. Ces caractères n'ont rien à voir avec des caractères dans l'écran. C'est seulement que l'éditeur hexadécimal considère chaque octet comme un code Ascii et affiche le caractère correspondant.

Contrairement à ce qui se passait avec l'écran texte, il n'est pas possible de visualiser tout l'écran HIRES d'un seul coup. En effet, alors que l'écran TEXT compte 28 lignes, l'écran HIRES en compte 200. Il faut donc faire défiler l'affichage de l'éditeur verticalement pour comparer les octets et caractères associés avec ce qu'on voit sur la recopie d'écran.

La figure ci-dessus, montre une telle tranche d'écran, et permet de comparer les octets (à gauche), le caractère Ascii correspondant (à droite) et la portion d'écran HIRES correspondante en dessous. Les caractères correspondant aux tirets des sprites sautent aux yeux au milieu de la forêt des caractères @.

Il "suffit" de noter (ou faire un collé/copié) des octets proprement dits (partie gauche de l'éditeur) pour avoir les data définissant chaque sprite. Enfin, c'était sans compter que les sprites se déplacent de façon très fluide (sur la figure ci-dessous, l'oiseau glisse, de un pixel à la fois, de gauche à droite dans l'écran). L'affichage d'un



sprite coïncide rarement avec les octets (sinon les déplacements seraient saccadés de 6 pixels en 6 pixels).

Dans le cas de cet écran Xenon1, les sprites font 18 pixels de large sur 18 pixels de haut. En théorie, il faut donc 3 tirets de 6 pixels pour afficher un sprite dans le sens de la largeur. En pratique, le programme en langage machine se charge de faire glisser le sprite dans une fenêtre de 4 tirets de large.

Par exemple, la première ligne du sprite situé en haut à gauche de l'écran (ligne qui inclut le haut des deux cornes) peut être définie par les pixels suivant (0=pixel éteint, 1=pixel visible) : 000000100001000000. Ce qui devrait correspondre aux 3 octets suivants (après avoir forcé le bit 6 à 1, drapeau indiquant qu'il ne s'agit pas d'un attribut) : #40, #61 et #40. Or on observe sur la figure que 4 octets sont impliqués : #40, #42, #44 et #40, ce qui, en ne conservant que les bits b0 à b5, correspond aux pixels suivants : 000000000010000100000000.

On voit que l'affichage du sprite de 18 pixels de large a été effectué sur 24 pixels. Pour récupérer les valeurs significatives des 3 octets il faudrait redécouper les 24 pixels en 3 zone de 6 pixel et laisser tomber (dans ce cas précis) les 4 pixels de tête et les 2 pixels de queue. En pratique c'est bien trop compliqué, à moins d'être un expert des tableurs genre Excel ou Calc !

La seconde méthode en pratique

La figure de gauche ci-dessus montre l'un des sprites de Xenon1, très agrandi. J'ai laissé volontairement une bande supplémentaire de 4 pixels de large à gauche et une autre de 2 pixels à droite pour que vous puissiez restituer et visualiser ce

b5	b4	b3	b2	b1	b0		
32	16	8	4	2	1		V
X					X	33	#21
	X			X		18	#12
	X	X	X	X		30	#1E
X		X	X		X	45	#2D
X	X	X	X	X	X	63	#3F
	X			X		18	#12
		X	X			12	#0C

que nous avions précédemment avec la méthode 1.

Résumons ce que l'on voit : On a un sprite de 18x18 pixels (3 tirets de 6 pixels de large sur 18 lignes HIRES de haut) affiché dans une zone de 24x18 pixels (4 tirets de 6 pixels de large sur 18 lignes HIRES).

Les lignes verticales blanches délimitent les octets directement impliqués dans le dessin du sprite. Chacune des 18 lignes HIRES doit donc être découpée en 3 tirets de 6 pixels de façon à récupérer le dessin réel du sprite. La nouvelle séparation en 3 zones de 6 pixels de large est visualisée par les lignes pointillées.

Au total, il nous faut donc numériser 18 fois 3 tirets pour récupérer tous les data du sprite. La figure à droite ci-dessus montre un exemple de numérisation. Il s'agit de la tête du sprite.

Le premier tiret par exemple (haut des cornes sur la tête) correspond à 100001 en binaire soit #21, valeur à laquelle il faudra ajouter #40 pour mettre le bit 6 à un (drapeau pour reconnaître les tirets), soit #61 dans cet exemple. Et ainsi de suite pour l'ensemble des 54 tirets du sprite.

Ceci représente un travail forcément important, mais moins qu'il n'y parait, car il y a beaucoup de redondances (tirets identiques).

A vous de voir si vous voulez vous aider soit d'un tableur (vous mettrez des croix dans les cases et il calcule les valeurs finales à utiliser), soit d'un simple tableau des 64 combinaisons possibles donnant directement les valeurs à utiliser.

Bon courage !