POUR QUELQUES DEFINITIONS DE PLUS...

Dans le courrier, les questions concernant les manipulations dans la saisie des listings reviennent très fréquemment, on va donc se faire un petit rappel de toutes les fonctions AMSTRAD CPC, dans ce domaine.

Pour saisir un listing (entrer au clavier les lignes Basic) tu passes en minuscules, tous les mots clé seront identifiés et transformés, automatiquement, en majuscules lors d'un LIST. La meilleure méthode, pour la numérotation des lignes, est d'utiliser la fontion AUTO. Tu composes:
AUTO 1ØØ,1Ø
L'éditeur affiche à l'écran le numéro de la ligne 1ØØ et, au prochain appui sur ENTER passera à la ligne 11Ø (saisie à partir de 1ØØ et intervalle des lignes 1Ø). A l'affichage du numéro, si celui-ci est suivi d'un astérisque (1ØØ*), ceci indique que la ligne contient déjà des instructions, donc prudence tu t'es peut-être trompé de numéro. A la fin d'une ligne Basic, constituée éventuellement de plusieurs lignes d'écran tu appuies sur ENTER.
Pour interrompre la saisie, avec la foction AUTO, tu appuies sur ESC (Break).
Dès que se termine ta saisie, tu sauvegardes ton listing par:
SAVE"breaker et ENTER
Si tu travailles avec une cassette, tu dois mettre la bande au début du support magnétique, sinon tu enregistres sur la bande amorce et, enfoncer les 2 touches REC et PLAY.
Tu lance l'exécution de ton programme par RUN s'il est déjà en mémoire ou, par
RUN"breaker
qui charge et lance le traitement.
Lorsque tu veux modifier ou, apporter un complément à ton programme:
- le charger en mémoire par: LOAD"breaker
- afficher la ligne à corriger par: EDIT num. ligne
- positionnner le curseur, par les touches flèches, sur la partie à modifier
- supprimer un caractère avec la touche CLR ou, composer le caractère à ajouter
- faire ENTER pour sortir de la ligne

L'éditeur Basic te permet, également, de copier le contenu d'une ligne vers une nouvelle ligne. Tu tiens la touche SHIFT et par les flèches tu déplaces un curseur bis. Lorsque ce second curseur atteint la zone à recopier, tu relaches les touches et par COPY tu effectues l'opération.

Il te reste encore à lister ton programme:
LIST te donne l'édition du début jusqu'à la fin, interruption par ESC.
LIST num. ligne affiche la ligne
LIST 2ØØ-35Ø édite les lignes 2ØØ à 35Ø

JE TE RECOIS 5 SUR 5....OVER....

Notre centre spatial vient de nous communiquer les derniers messages de détresse.
Alexandre FERTAY vient, lui aussi, de chuter sur la ligne 96Ø de JOYSTICK-ATTACK, dans laquelle on oublie souvent de composer la virgule:
96Ø LOCATE posx(j),posy(j)
UnE aventurièrE!! Génial!! **Pascale BOUCHET** dans ATTACK passe et repasse en GAME OVER sans pouvoir jouer. Le programme se branche sur la fin de partie si le nombre de "vies" est égal à Ø. Tu as, sans doute, initialisé la variable "vie" et tu testes "vies" ou l'inverse. Controle bien les lignes 184Ø et 25Ø.
Le rappel de "POUR QUELQUES DEFINITIONS DE PLUS" tombe à pic pour **Laurent BREUGNOT** et ses soucis de sauvegarde.

R.A.S plus rien à signaler, ne t'inqiète pas, j'assure la veille 24h sur 24.

JOYSTICK EN KIT

Je t'annonçais, la semaine dernière, une initiation suivant un mode nouveau. Le principe est simple, le programme qui sert de base aux explications et commentaires de ta rubrique "J'APPRENDS" sera TON PROGRAMME, sélectionné parmi tous les listings que nous recevons. Au passage, salut à **Sylvain KOUBDJANIAN** qui vient de nous adresser deux jeux qu'il suffirait d'améliorer un peu en vitesse pour pouvoir les utiliser. Pour être retenu, tu dois faire très fort, avec une bonne idée de base et, des animations rapides. Et tout celà en Basic? difficile c'est vrai. Pour y arriver je te propose un ensemble de routines que tu vas intégrer dans tes listings, un véritable KIT à assembler suivant tes besoins. Avec celà tu auras toutes les chances de voir ton programme retenu soit pour servir de base à J'APPRENDS, soit dans la nouvelle rubrique PROGRAMING. La plus haute marche du podium t'attend!!!
Avant tout, je voudrais te montrer très rapidement les limites de l'animation dans un programme entièrement en Basic. Le mini programme est très simple, un avion, commandé au joystick, monte ou descend devant un ciel uniforme et tire. Le nombre de tirs en cours possibles est limité à 1Ø. Le déplacement du projectile s'accélère progressivement. Avec un seul tir en cours le délai de réponse passe bien mais, dès que tu multiplies le tir les temps s'écroulent.
Tu saisis le listing et tu testes les temps. L'avion est tout simplement constitué de 5 caractères, sans idée de dessin, il s'agit d'un essai et non d'un exercice graphique.

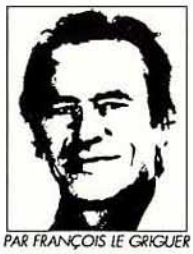
```
1Ø 'avion et tir en basic
2Ø '
3Ø MODE 1:INK Ø,11
4Ø avion$=CHR$(197)+CHR$(154)+CHR$(159)+
CHR$(154)+CHR$(243)
5Ø nbf=Ø:DIM spr(1Ø,3)
6Ø '
7Ø 'dessin decor
8Ø '
9Ø FOR y=Ø TO 143 STEP 2
1ØØ MOVE Ø,y:DRAWR 639,Ø,1:NEXT
11Ø FOR y=y TO 399 STEP 2
12Ø MOVE Ø,y:DRAWR 639,Ø,Ø:NEXT
13Ø xa=2:ya=16
14Ø '
15Ø GOSUB 43Ø
16Ø mdf=JOY(Ø):IF mdf=Ø THEN 15Ø
17Ø IF mdf>15 THEN 34Ø
18Ø IF mdf=2 THEN 22Ø
19Ø IF mdf=1 THEN 28Ø
2ØØ GOTO 15Ø
21Ø '
22Ø 'montee avion
23Ø '
24Ø IF ya=1 THEN 15Ø
25Ø LOCATE xa,ya:PEN Ø:PRINT STRING$(5,143)
26Ø ya=ya-1:LOCATE xa,ya:PEN 3:PRINT avion$:
GOTO 15Ø
27Ø '
28Ø 'descente
29Ø '
3ØØ IF ya=16 THEN 15Ø
31Ø LOCATE xa,ya:PEN Ø:PRINT STRING$(5,143)
32Ø ya=ya+1:LOCATE xa,ya:PEN 3:PRINT avion$:
GOTO 15Ø
33Ø '
34Ø ' fire
35Ø '
36Ø IF nbf=1Ø THEN 4ØØ
37Ø nbf=nbf+1
38Ø FOR j=1 TO 1Ø:IF spr(j,1)=Ø THEN 39Ø ELSE NEXT
39Ø spr(j,1)=xa+5:spr(j,2)=ya:spr(j,3)=1
4ØØ mdf=mdf-16
41Ø IF mdf<16 THEN 17Ø ELSE mdf=mdf-16:GOTO 41Ø
42Ø '
43Ø 'afficher tir
44Ø '
45Ø FOR j=1 TO 1Ø
46Ø xf=spr(j,1):yf=spr(j,2):IF xf=Ø THEN 53Ø
47Ø LOCATE xf,yf:PEN Ø:PRINT CHR$(143):PEN 3
48Ø xf=xf+spr(j,3)
49Ø IF xf>4Ø THEN nbf=nbf-1:spr(j,1)=Ø:GOTO 53Ø
5ØØ LOCATE xf,yf:PRINT CHR$(246);
51Ø spr(j,1)=xf:IF spr(j,3)>7 THEN 53Ø
52Ø spr(j,3)=spr(j,3)+1
53Ø NEXT j:RETURN
```

Imagine d'ajouter des adversaires qui tirent eux aussi, un mini décor au sol qui se déplace, pas évident, voire impossible. Cette animation va nous servir pour voir dans le détail les routines de notre KIT et l'accélération du traitement. Un super programme en perspective!!! Ne manque surtout pas ce nouveau départ.

Bien joystiquement vôtre
François LE GRIGUER

LA SEMAINE PROCHAINE

Le début des routines de JOYSTICK EN KIT, du Basic toujours mais aussi de l'ASSEMBLEUR à la portée de tous.



PAR FRANÇOIS LE GRIGUER

JOYSTICK EN KIT

TU VIENS DE TE POSER EN DOUCEUR SUR UNE NOUVELLE PLANETE DE L'INFORMATIQUE. UNE PLANETE DIFFERENTE DES AUTRES PAR UNE UTILISATION REGULIERE DE L'ASSEMBLEUR. LES PROGRAMMES ECRITS ICI, SONT TOUJOURS EN BASIC POUR L'ORGANISATION GENERALE MAIS, CHAQUE SEQUENCE JUGEE TROP LENTE SE TROUVE REMPLACEE PAR DE L'ASSEMBLEUR ET, TOUT CELA, DANS UN BUT, UN SEUL, ALLER VITE, TRES VITE, AVEC UNE ANIMATION FLUIDE. CES SOUS-PROGRAMMES EN ASSEMBLEUR VONT TE PERMETTRE DE CREER DES JEUX RAPIDES AVEC PLUSIEURS SPRITES ANIMES ET, NOUS ALLONS VOIR, ENSEMBLE, LEUR CONTENU ET LEUR UTILISATION A TRAVERS LA REALISATION D'UN JEU D'ARCADE.

SOMMAIRE

DESSINE-MOI UN AVION

Tu as vu la semaine dernière, dans le petit programme en Basic, un assemblage de caractères qui occupaient la place d'un avion. Le but n'était pas de dessiner mais de controler les limites d'une séquence écrite en Basic. Mais je te devais un vrai dessin qui, en même temps, nous donne l'occasion d'étudier le principe de préparation d'un Sprite, c-à-d d'un objet destiné à être déplacé (Sprite en français veut dire lutin).
Les méthodes varient, la plus simple consiste à juxtaposer des carcatères redéfinis et, la plus complexe décompose le dessin en octets codifiés suivant les couleurs. L'affichage de carcatères, malgré un temps de traitement plus long, permet une meilleure approche et, c'est le processus que l'on va retenir. Le gros défaut des caractères étant un affichage en une seule couleur, nous, nous allons prévoir la superposition de couleurs pour obtenir un beau Sprite, en assembleur évidemment pour être rapide.
Tu prends du papier quadrillé 5/5 et tu représentes des grands carrés de 8 par 8. Chaque carré constitue un caractère et, ceci te permet de définir les dimensions de ton Sprite, en longueur et en hauteur. L'écran en mode 1 contient 25 lignes de 40 caractères. Tu traces les contours de ton objet, puis les détails internes, en respectant chaque carreau qui constitue un pixel (point). Maintenant, avec ton dessin entièrement tracé, tu colories en sachant que, en mode 1, tu disposes de 4 couleurs. C'est exactement la présentation de l'avion avec, 6 caractères de long et 3 lignes, les INK matérialisées par le code 1, 2 ou 3.
Tu constates, maintenant, que plusieurs caractères contiennent plus d'une couleur. Pour reproduire un caractère comme CX (colonne C ligne X), par exemple, il te faudra afficher un caractère contenant tous les pixels 2 avec l'INK 2 et, superposer un second pour tous les pixels 1 avec l'INK 1, d'ou 2 caractéres à redéfinir pour afficher CX à l'écran avec ses deux couleurs. Toute cette décomposition génère finalement 25 caractères à codifier pour être redéfinis.
Ton avion se précise, il te reste encore à calculer, pour chacune des 8 lignes des 25 caractères, la valeur représentée par les pixels.

Avec un petit rappel de l'octet et de l'hexadécimal tu vas voir: c'est Super Simple!!!
L'octet, avec ses 8 bits, le voici:

128 64 32 16 8 4 2 1

Le bit 0 vaut 1, le bit 1 vaut 2 etc... jusqu'au bit 7 qui vaut 128.

Si tu prends la première ligne du caractère AX, tu as 4 pixels, les bits 7,6,5 et 4, qui donnent en décimal: 128 + 64 + 32 + 16 = 240. Cette addition s'avère fastidueuse et, il serait beaucoup plus simple de dire: 8 + 4 + 2 + 1 = 15. Génial non!!! tu viens d'utiliser l'hexadécimal en divisant par 16 toutes les valeurs décimales et, tu écris non plus = 15 mais = F. En hexadécimal 10, 11, 12, 13, 14 et 15 sont remplacés par A, B, C, D, E et F.

Notre octet se présente maintenant avec les valeurs hexadécimales:

8 4 2 1 8 4 2 1

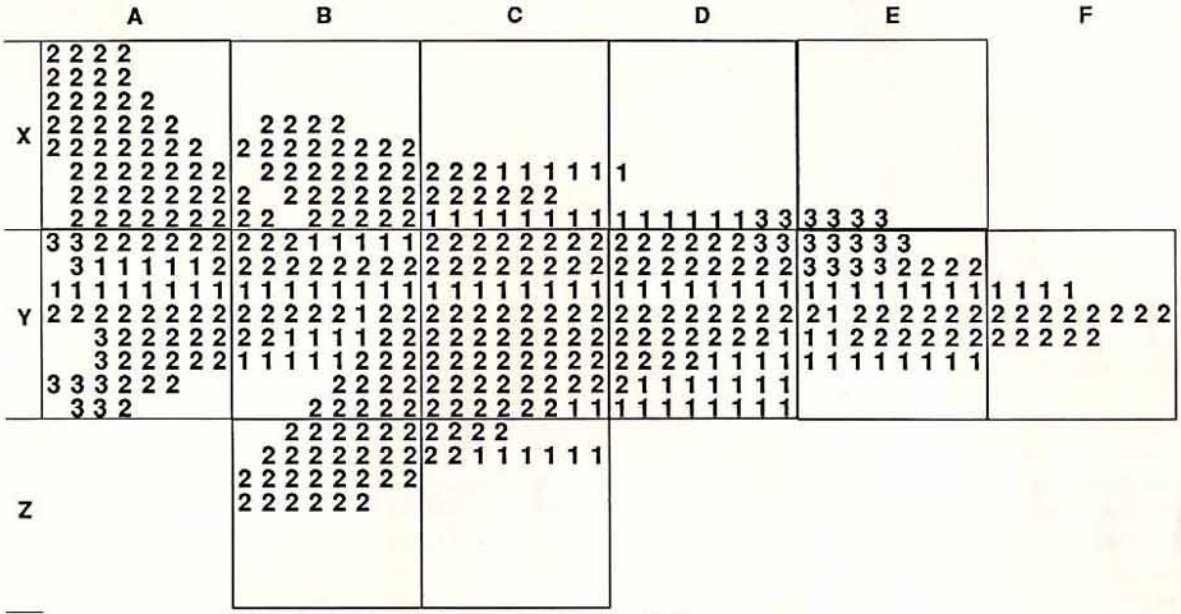
Et un petit exemple avec la dernière ligne du caractère AX, avec les bits 0, 1, 2, 3, pour la partie basse qui donne: 1 + 2 + 4 + 8 = F (15). La partie haute, avec les bits 4, 5, 6 = 1 + 2 + 4 = 7. Finalement la valeur totale de l'octet s'écrit: 7F soit 127 en décimal (7 * 16 = 112 + 15).
Maintenant tu prends du papier et un crayon et tu codifies les 25 caractères, sans regarder la solution!! on ne copie pas là-bas au fond!! attention j'ai les noms!!
Dès que tu as rendu ta feuille on met des notes sur le petit controle en base 16 et, il faut avoir presque 20/20!! Le programme qui suit contient toute la redéfinition des caractères avec l'instruction SYMBOL et, l'affichage pour controle, de ton Super avion. Un avion qui va, très bientot s'animer. Comme toujours, tu saisis le listing avec en début AUTO 10,10, et en fin SAVE»KIT. Un petit RUN pour découvrir ton dessin en changeant les couleurs suivant tes gouts en ligne 580. Le symbole & qui précéde toutes les valeurs signifie qu'il s'agit d'hexadécimal.

```
10 'definition des caracteres
20 SYMBOL AFTER 150
30 '
40 'avion
50 '
60 'AX INK2
70 SYMBOL 255,&F0,&F0,&F8,&FC,&FE,&7F,&7F,&7F
80 'BX INK 2
90 SYMBOL 254,0,0,0,&78,&FF,&7F,&BF,&DF
100 'CX INK 1
110 SYMBOL 253,0,0,0,0,0,&1F,0,&FF
120 'CX INK 2
130 SYMBOL 252,0,0,0,0,0,&E0,&FC
140 'DX INK 1
150 SYMBOL 251,0,0,0,0,0,&80,0,&FC
160 'DX INK 3
170 SYMBOL 250,0,0,0,0,0,0,0,0,&3
180 'EX INK 3
190 SYMBOL 249,0,0,0,0,0,0,0,0,&F0
200 'AY INK 1
210 SYMBOL 248,0,&3E,&FF
220 'AY INK 2
230 SYMBOL 247,&3F,&1,0,&FF,&1F,&1F,&1C,&10
240 'AY INK 3
250 SYMBOL 246,&C0,&40,0,0,&20,&20,&E0,&60
260 'BY INK 1
270 SYMBOL 245,&1F,0,&FF,&4,&3C,&F8
280 'BY INK 2
290 SYMBOL 244,&E0,&FF,0,&FB,&C3,&7,&F,&1F
300 'CY INK 1
310 SYMBOL 243,0,0,&FF
320 'CY INK 2
330 SYMBOL 242,&FF,&FF,0,&FF,&FF,&FF,&FF,&FC
340 'DY INK 1
350 SYMBOL 241,0,0,&FF,0,&1,&F,&7F,&FF
360 'DY INK 2
370 SYMBOL 240,&FC,&FF,0,&FF,&FE,&F0,&80
380 'DY INK 3
390 SYMBOL 239,&3
400 'EY INK 1
410 SYMBOL 238,0,0,&FF,&40,&C0,&FF
420 'EY INK 2
430 SYMBOL 237,0,&F,0,&BF,&3F
440 'EY INK 3
450 SYMBOL 236,&F8,&F0
460 'FY INK 1
470 SYMBOL 235,0,0,&F0
480 'FY INK 2
490 SYMBOL 234,0,0,0,&FF,&F8
500 'BZ INK 2
510 SYMBOL 233,&3F,&7F,&FF,&FC
520 'CZ INK 1
530 SYMBOL 232,0,&3F
540 'CZ INK 2
550 SYMBOL 231,&F0,&C0
```

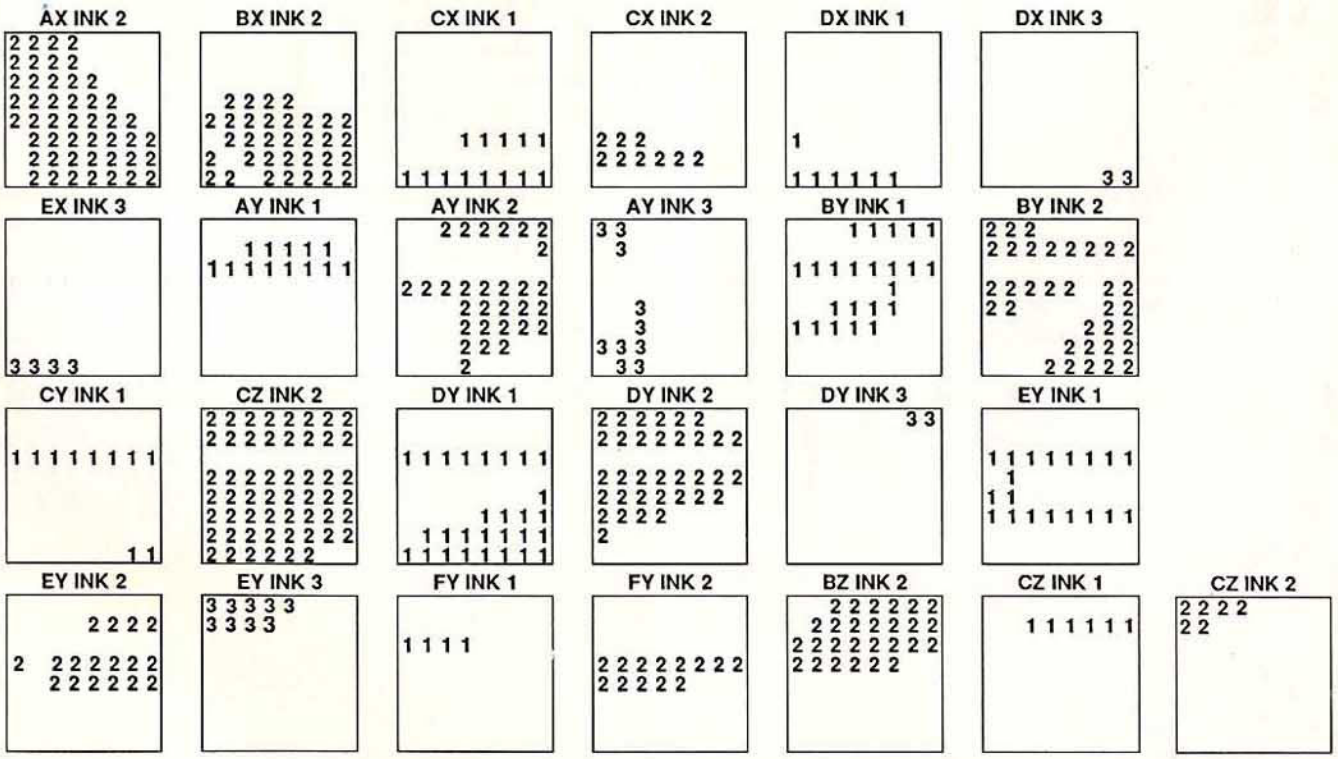
```
56Ø '
57Ø 'edition pour controle
58Ø MODE 1:x=1Ø:y=1Ø:INK Ø,Ø:INK 1,13:INK 2,9:INK 3,7
59Ø PRINT CHR$(22)+CHR$(1);
6ØØ LOCATE x,y:PEN 2
61Ø PRINT CHR$(255);CHR$(254);
62Ø PEN 1:PRINT CHR$(253);CHR$(8);
63Ø PEN 2:PRINT CHR$(252);
64Ø PEN 1:PRINT CHR$(251);CHR$(8);
65Ø PEN 3:PRINT CHR$(25Ø);CHR$(249);
66Ø LOCATE x,y+1
67Ø PEN 1:PRINT CHR$(248);CHR$(8);
68Ø PEN 2:PRINT CHR$(247);CHR$(8);
69Ø PEN 3:PRINT CHR$(246);
7ØØ PEN 1:PRINT CHR$(245);CHR$(8);
71Ø PEN 2:PRINT CHR$(244);
72Ø PEN 1:PRINT CHR$(243);CHR$(8):
73Ø PEN 2:PRINT CHR$(242);
74Ø PEN 1:PRINT CHR$(241);CHR$(8);
75Ø PEN 2:PRINT CHR$(24Ø);CHR$(8);
76Ø PEN 3:PRINT CHR$(239);
77Ø PEN 1:PRINT CHR$(238);CHR$(8);
78Ø PEN 2:PRINT CHR$(237);CHR$(8);
79Ø PEN 3:PRINT CHR$(236);
8ØØ PEN 1:PRINT CHR$(235);CHR$(8);
81Ø PEN 2:PRINT CHR$(234);
82Ø LOCATE x+1,y+2:PRINT CHR$(233);
83Ø PEN 1:PRINT CHR$(232);CHR$(8);
84Ø PEN 2:PRINT CHR$(231);
```

Cette séquence d'affichage est là uniquement pour te permettre de voir le dessin mais, elle sera remplacée par un sous-programme en assembleur. Et, quand on en sera à l'animation, tu ajouteras derrière l'appareil un échappement de tuyère variable, comme dans un jeu de professionnel!!! Voilà pour le premier Sprite, il y en aura d'autres, après avoir vu comment les organiser.

DESSIN D'UN AVION PAR CARACTERES REDEFINIS



DECOMPOSITION DE L'AVION EN CARACTERES



La semaine prochaine tu auras une vue d'ensemble de la gestion d'un Sprite, avec le détail du contenu des descripteurs. Ensuite ce sont les routines en assembleur et, les échanges avec le Basic. Tout un ensemble pour nous préparer des listings d'enfer!!!

Bien joystiquement vôtre
François LE GRIGUER

LA SEMAINE PROCHAINE

L'organisation complète de la gestion d'un Sprite et le début des routines en Assembleur. Tous les commentaires pour l'Assembleur à la portée de tous.

PAR FRANÇOIS LE GRIGUER

JOYSTICK EN KIT

DES SPRITES AU PROGRAMME

TU AS VU L'AVION!!! IL VA ETRE BIENTOT TRES REDOUTABLE TON AVION, TIR DE ROQUETTES, MISSILES A TETE CHERCHEUSE, BOMBES, TOUT UN ARSENAL DE COMBAT POUR DETRUIRE LES ADVERSAIRES LES PLUS DIVERS, EN L'AIR ET AU SOL, ET TOUT CELA EN DECOUVRANT UNE GESTION DE SPRITES SIMPLE ET BIEN ORGANISEE. UNE DOSE DE BASIC PLUS DEUX DOSES D'ASSEMBLEUR, TU SECOUES BIEN L'ENSEMBLE POUR OBTENIR UN SUPER COCKTAIL RAPIDE, EFFICACE, EXPLOSIF!!! ATTENTION LES YEUX!!!

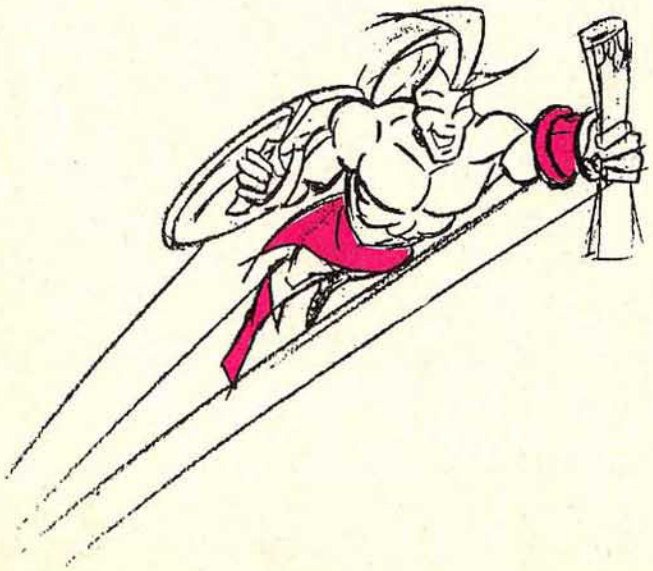
J'APPRENDS

Pour dessiner des engins ou des personnages tu peux t'inspirer des B.D, et faire le découpage et la codification des caractères ne pose pas de problèmes majeurs. Par contre toute l'organisation nécessaire à une gestion rapide demande un peu plus de réflexion avec, deux ensembles complémentaires l'un de l'autre, le fichier des données et celui des descripteurs.

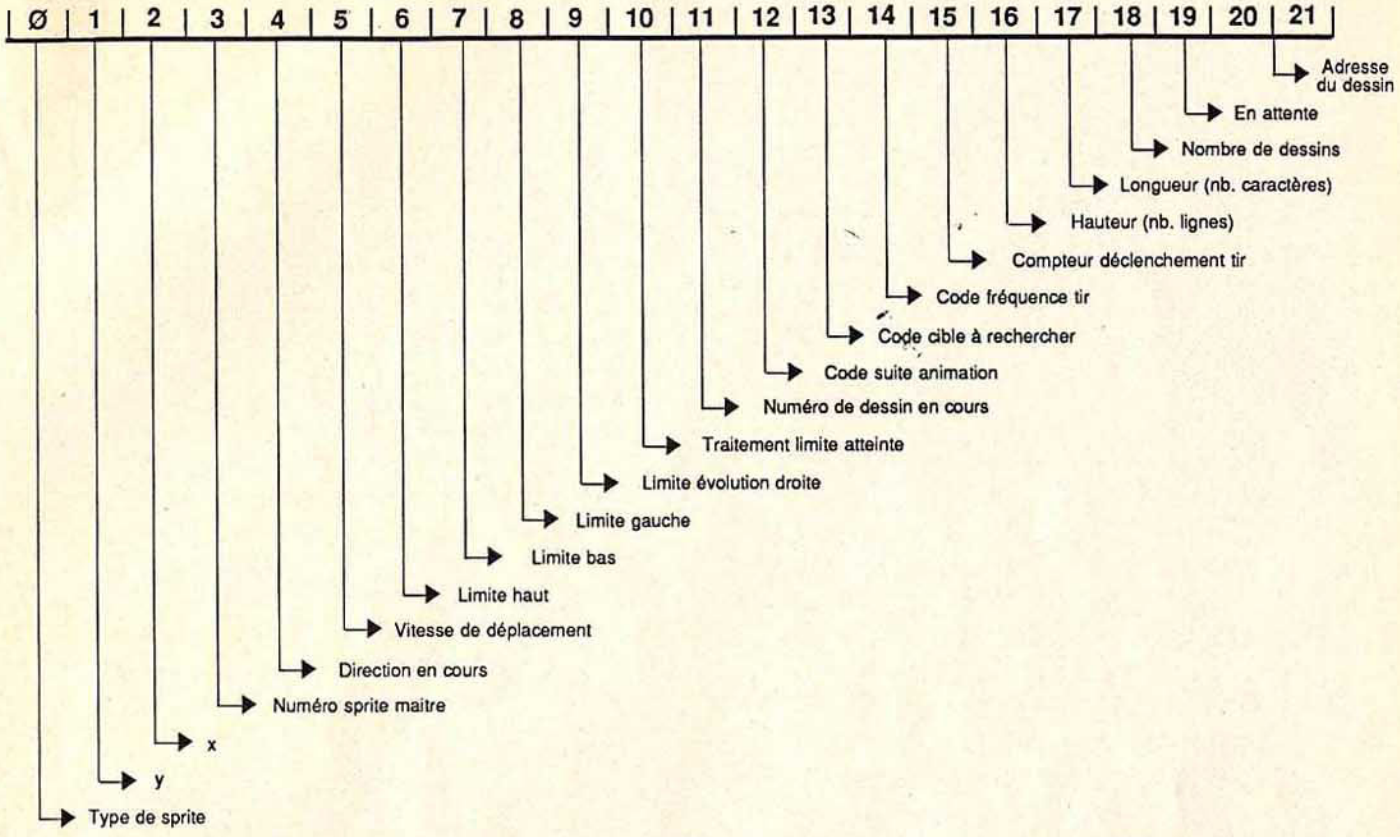
Les descripteurs contiennent tous les codes servant à l'identification et à la gestion du Sprite. Les données servent uniquement à l'affichage des caractères qui constituent le Sprite. Ce que nous allons définir te permettra d'animer n'importe quel type de Sprite.

L'objectif, avec le fichier des données, c'est de pouvoir afficher à l'écran un Sprite en suivant les lignes, les caractères et les couleurs prévus. De plus il pourra être animé, c-à-d, constitué de plusieurs dessins, par exemple l'échappement de notre avion va s'animer à l'aide d'au moins trois images différentes. Une animation doit comporter au minimum trois images, avec deux cela donne un simple clignotement. Pour un personnage qui marche, ce nombre sera au moins de quatre ou cinq. Donc le fichier des données contient une suite d'octets codifiés suivant les INK 1,2 et 3. Comment va-t-on l'obtenir? par l'intermédiaire d'un programme: le générateur de sprites que l'on verra en détails. Dans l'immédiat c'est le descripteur du sprite qui nous intéresse.

J'ai appelé descripteur l'ensemble des codes qui vont nous permettre de gérer un sprite et ceci, quel que soit son type. L'organisation que je te propose n'a rien de standard et pourrait être étendue, ou prévue de façon complètement différente, mais elle répond à un minimum de besoins. Le plan du descripteur te donne le contenu de chacun des 22 octets. Ces 22 informations, vont intervenir dans la gestion du sprite suivant le role et l'utilisation définie. Cette analyse de chaque code, du sens à donner aux différentes valeurs, est à étudier avec beaucoup de soins, et, te montre, dans le détail, comment construire un programme. L'analyse est indispensable, qu'il s'agisse d'un jeu ou d'une toute autre application. Ce plan te servira sur n'importe quel matériel.



L'IDENTIFICATEUR D'UN SPRITE



Le Type de Sprite, en octet Ø, correspond à 11 codes:
1 = engin joueur. 2 = une explosion
3, 4 et 5 = 3 types d'adversaires
6, 7 et 8 = 3 types de tir joueur, par exemple roquette, missile et bombe
9, 1Ø et 11 = 3 types de tir adversaire
Le traitement ajoutera 128 au type, pour marquer un sprite touché en cours d'explosion.

Les octets 1 et 2 gèrent les coordonnées x et y en caractères (x = 1 à 4Ø et y = 1 à 25 origine haut gauche en écran) du coin haut gauche du sprite. Un sprite inactif contient Ø en x.

Le code sprite maitre, en octet 3, lie le sprite à un autre et, les coordonnées xy sont alors relatives (à ajouter aux coordonnées du sprite maitre pour calculer xy de ce sprite). A utiliser, par exemple, pour ajouter à notre avion un échappement animé sans avoir besoin de dessiner plusieurs fois l'avion en totalité.

L'octet 4, avec la direction en cours, associé à la vitesse, donne la valeur de mise à jour des coordonnées, dans les déplacements. Une table par vitesse, contient, dans chaque direction la valeur + - pour x et pour y. 8 directions prévues de 1 à 8, verticales, horizontales et obliques. Par le choix de la direction et de la vitesse, les sprites évoluent de manière très variée.

La vitesse, en octet 5, est prévue avec 3 vitesses 1, 2 ou 3

Les limites d'évolution du sprite, octets 6, 7, 8, 9, donnent au programme tous les éléments pour tester la progression. Lorsqu'une limite est atteinte le code en octet 1Ø, décide de la suite à exécuter.

Le code octet 1Ø, indique la suite à donner au sprite en atteinte de limite d'évolution. Ø = le supprimer (mettre Ø en x). 4 = inverser la direction en cours. Tout autre valeur conserve le sprite avec Ø en direction (sans déplacement)

Le numéro de dessin en cours, octet 11, concerne l'affichage et progresse de 1 à chaque passage, ceci pour animer un mouvement. Il est lié au nombre de dessins en octet 18. Si le numéro en cours est > que le nombre, le numéro en cours revient automatiquement à 1, ou le sprite disparait, tout dépend du code en octet 12. Une opération très pratique pour un évènement momentané comme une explosion qui doit disparaitre à la fin de son animation.

L'octet 12 avec Ø élimine le sprite en fin d'animation, sinon retour au dessin numéro 1.

Si une cible particulière est visée, l'octet 13 donne le numéro du sprite et, la mise à jour des coordonnées tient compte de la position à atteindre. A utiliser pour des missiles téléguidés.

La fréquence de tir d'un adversaire à l'autre varie pour éviter toute monotonie et, l'octet 14 contient la valeur de la fréquence.

Le compteur de déclenchement du tir, octet 15, incrémenté de 1 à chaque passage, initialise le tir dès qu'il atteint la valeur de la fréquence.

En octet 16, il s'agit de la hauteur totale du sprite en nombre de lignes de caractères (pour l'avion nombre de lignes = 3). Indispensable à l'affichage et utilisé en gestion des collisions pour connaitre la taille du sprite Ce nombre est converti par le programme en lignes graphiques (1 ligne caractère = 8 lignes graphiques).

L'octet 17 complète l'octet 16, avec la largeur, en caractères, du sprite (en mode 1 le caractère correspond, en largeur, à 2 octets).

L'octet 19 reste en attente pour un code à venir.

Les deux derniers octets, 2Ø et 21, assurent le lien entre le fichier identificateur et le fichier des données avec, l'adresse en données du premier dessin. Pour accéder aux dessins suivants, dans le cas d'animation, le programme ajoute à l'adresse du premier, le nombre d'octets d'un dessin (nombre de caractères multiplié par 2 octets, multiplié par 8 lignes graphiques et, multiplié par le nombre de lignes).

Le descripteur fonctionne, sur le papier actuellement, et va se concrétiser avec le générateur de sprites. Un programme qui assure la conversion des caractères redéfinis vers les octets en données et, le transfert des descripteurs décrits en DATA. Un système extrèmement simple pour toi, je te le garantis!!!
Et l'assembleur dans tout celà!!! Ne t'inqiète pas, il arrive ensuite pour gérer les fichiers des sprites et là, la vitesse... d'ENFER!!!
Avec le descripteur de sprite que tu viens de voir, je suis certain que tu nous prépares des scénarios démoniaques, des listings qui décoiffent!!!

Bien joystiquement vôtre
François LE GRIGUER

LA SEMAINE PROCHAINE

Le GENERATEUR DE SPIRITES avec tous les commentaires

PAR FRANÇOIS LE GRIGUER

J'APPRENDS

SALUT A TOUS LES FOUS DE PROGRAMMATION. LES SPRITES, APRES LA DEFINITION DES DESCRIPTEURS DE LA SEMAINE DERNIERE, DECOLLENT, AUJOURD'HUI, AVEC LA CONVERSION DE TES DESSINS EN OCTETS POUR LA MISE A JOUR DU FICHIER DES DONNEES. TOUT CELA, DANS LE LISTING DU GENERATEUR DE SPRITES, DOCUMENTE AU MAXIMUM, POUR T'AIDER A TOUT COMPRENDRE.

FLASHBACK SUR JOYSTICK-BREAKER

Beaucoup d'aventuriers se posent des questions à propos du fichier "Tableaux" et un petit retour en arrière s'impose. JOYSTICK-BREAKER est constitué d'un programme et, d'un fichier de données "Tableaux" contenant la description du contenu des 4Ø table<aux, suivant ta redéfinition. Lors du premier RUN que tu lances, ce fichier qui n'a pas encore été mis à jour par le programme, doit exister sur la disquette ou sur la cassette, pour permettre le LOAD de la ligne 26Ø. Comme je te l'expliquais dans JOYSTICK Hebdo numéro 7 à la page 15, tu crées "Tableaux" en composant directement à l'écran:
POKE 33ØØØ,Ø.. SAVE "tableaux", B,33ØØØ,1
Ceci enregistre le fichier "tableaux" en binaire, avec une longueur d'un octet qui contient Ø (nombre de tableaux en cours = Ø). Un dernièr rappel, si tu travailles avec une cassette, tu dois impérativement en utiliser deux: l'une pour le programme et l'autre pour le fichier "tableaux". Ce fichier sera, ensuite, géré automatiquement par le programme. OK!! ça marche!! Bon BREAKER à tous!!!

JOYSTICK EN KIT

UN GENERATEUR DE SPIRITES

Tu as préparé ton dessin, décomposé en caractères et, codifié les lignes. Ce générateur doit, maintenant, afficher le sprite et, mettre à jour le fichier des données en recopiant les octets de la mémoire écran. Pourquoi ce procédé? pour rester simple et, t'éviter d'avoir à codifier chaque octet en fonction des pixels et de leur INK, ce qui s'avère assez fastidueux. L'autre fonction de ce programme consiste à générer le fichier des identificateurs à partir de DATA.
Le listing contient une masse de commentaires en italique qui te permettent de mieux comprendre le BASIC mais, tu ne dois surtout pas les saisir dans ton listing. Compose: AUTO 1Ø,1Ø pour une numérotation automatique des lignes. Certaines lignes étant coupées pour des raisons de mise en page, tu ne fais ENTER qu'avant un nouveau numéro de ligne.

```
1Ø **********************************
2Ø '       GENSPRIT
3Ø **********************************
4Ø ' redefinition des caracteres
5Ø ' data identificateur sprite
6Ø ' afficher le sprite
7Ø ' transfert des octets sprite en donnees
8Ø ' transfert de l'identificateur en fichier sprite
9Ø '-------------------------
1ØØ '
11Ø ' adrid=zone identificateurs sprites
12Ø ' adrdata=zone donnees sprites
13Ø ' nbspr=nombre de sprites declares
14Ø '
15Ø SYMBOL AFTER 15Ø
16Ø MEMORY 35999:adrid=36ØØØ:adridØ=adrid
17Ø nbspr=2
Cette valeur est à adapter en fontion du nombre de sprites prévus.
18Ø adrdata=adrid+(nbspr*22)
Calcul de la place nécessaire à la partie descripteur.
19Ø ' redefinition des caracteres
Toute la redéfinition a déjà été vue, elle n'est donnée qu'à titre de rappel.
2ØØ '
21Ø 'avion
22Ø '
23Ø 'AX INK2
24Ø SYMBOL 255,&FØ,&FØ,&F8,&FC,&FE,&7F,&7F,&7F
25Ø 'BX INK 2
26Ø SYMBOL 254,Ø,Ø,Ø,&78,&FF,&7F,&BF,&DF
27Ø 'CX INK 1
28Ø SYMBOL 253,Ø,Ø,Ø,Ø,Ø,&1F,Ø,&FF
29Ø 'CX INK 2
3ØØ SYMBOL 252,Ø,Ø,Ø,Ø,Ø,&EØ,&FC
31Ø 'DX INK 1
32Ø SYMBOL 251,Ø,Ø,Ø,Ø,Ø,&8Ø,Ø,&FC
33Ø 'DX INK 3
34Ø SYMBOL 25Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,&3
35Ø 'EX INK 3
36Ø SYMBOL 249,Ø,Ø,Ø,Ø,Ø,Ø,Ø,&FØ
37Ø 'AY INK 1
38Ø SYMBOL 248,Ø,&3E,&FF
39Ø 'AY INK 2
4ØØ SYMBOL 247,&3F,&1,Ø,&FF,&1F,&1F,&1C,&1Ø
41Ø 'AY INK 3
42Ø SYMBOL 246,&CØ,&4Ø,Ø,Ø,&2Ø,&2Ø,&EØ,&6Ø
43Ø 'BY INK 1
44Ø SYMBOL 245,&1F,Ø,&FF,&4,&3C,&F8
45Ø 'BY INK 2
46Ø SYMBOL 244,&EØ,&FF,Ø,&FB,&C3,&7,&F,&1F
47Ø 'CY INK 1
48Ø SYMBOL 243,Ø,Ø,&FF
49Ø 'CY INK 2
5ØØ SYMBOL 242,&FF,&FF,Ø,&FF,&FF,&FF,&FF,&FC
51Ø 'DY INK 1
52Ø SYMBOL 241,Ø,Ø,&FF,Ø,&1,&F,&7F,&FF
53Ø 'DY INK 2
54Ø SYMBOL 24Ø,&FC,&FF,Ø,&FF,&FE,&FØ,&8Ø
55Ø 'DY INK 3
56Ø SYMBOL 239,&3
57Ø 'EY INK 1
58Ø SYMBOL 238,Ø,Ø,&FF,&4Ø,&CØ,&FF
59Ø 'EY INK 2
6ØØ SYMBOL 237,Ø,&F,Ø,&BF,&3F
61Ø 'EY INK 3
62Ø SYMBOL 236,&F8,&FØ
63Ø 'FY INK 1
64Ø SYMBOL 235,Ø,Ø,&FØ
65Ø 'FY INK 2
66Ø SYMBOL 234,Ø,Ø,Ø,&FF,&F8
67Ø 'BZ INK 2
68Ø SYMBOL 233,&3F,&7F,&FF,&FC
69Ø 'CZ INK 1
7ØØ SYMBOL 232,Ø,&3F
71Ø 'CZ INK 2
72Ø SYMBOL 231,&FØ,&CØ
73Ø ' ------------------------
74Ø '  DATA des descripteurs sprites
75Ø ' ------------------------
76Ø ' descripteur avion joueur
77Ø DATA 1,2,15,Ø,1,1,1,16,1,4,4,1,1,Ø,Ø,Ø,3,6,1,Ø
Pour bien interpréter le contenu, il est indispensable de reprendre l'explication du descripteur de la semaine dernière. Les 2Ø premiers octets sont prévus en DATA. Pour les 2 derniers, l'adresse en données, le programme s'en charge.
78Ø 'numero sequence affichage autres dessins
79Ø DATA Ø,Ø,Ø
Utilisé pour des sprites animés (plusieurs dessins).
8ØØ '
81Ø ' descripteur adversaire
82Ø DATA 3,39,2,Ø,6,2,1,19,2,4Ø,4,1,1,Ø,Ø,Ø,1,1,2,Ø
83Ø 'numero sequence affichage autres dessins
84Ø DATA 1,Ø,Ø
Ici l'adversaire contient 1 second dessin, Il est donc animé. Le nombre d'adversaires augmentera, dans l'immédiat un seul nous
```

```
suffit pour comprendre le processus.
85Ø ' ------------------------
86Ø ' mise a jour du fichier des sprites
87Ø ' ------------------------
88Ø FOR j=1 TO nbspr
Début de la boucle de programme suivant le nombre de sprites.
89Ø MODE 1:x=1:y=1:INK Ø,Ø:INK 1,13:INK 2,9:INK 3,7
9ØØ ON j GOSUB 151Ø,185Ø
Branchement dans la séquence d'affichage des caractères du sprite.
Si J=1 exécution du sous-programme en ligne 151Ø, avec 2 en ligne
185Ø.
91Ø '------------------------
92Ø ' transfert des DATA sprites
93Ø '------------------------
94Ø '
Après l'affichage du sprite à l'écran, le programme met à jour les
fichiers en commençant par la partie identificateur décrite en DATA.
95Ø ' transfert des 2Ø octets identificateur
96Ø FOR k=Ø TO 19
97Ø READ a:POKE adrid+k,a:NEXT k
98Ø ' definition adresse donnees
99Ø h=Ø:adrdata2=adrdata
1ØØØ IF adrdata2>32768 THEN h=h+128:
adrdata2=adrdata2-32768
1Ø1Ø IF adrdata2>1Ø23 THEN h=h+4:
adrdata2=adrdata2-1Ø24:GOTO 1Ø1Ø
1Ø2Ø IF adrdata2>255 THEN h=h+1:
adrdata2=adrdata2-256:GOTO 1Ø2Ø
1Ø3Ø POKE adrid+2Ø,adrdata2:POKE adrid+21,h
Il s'agit de convertir l'adresse "adrdata" en 2 octets qui seront
interprétés par l'Assembleur (octets 2Ø et 21 du descripteur). Le mi-
croprocesseur Z8Ø gère le premier octet en poids faible de l'adresse,
valeur inférieure à 256 et, le second octet en poids fort, valeur
supérieure à 255. Une adresse tient sur 16 bits (2 octets) et la valeur
de chaque bit devient alors en décimal:
32768 16384 8192 4Ø96 2Ø48 1Ø24 512 256
128 64 32 16 8 4 2 1
1Ø4Ø nboct=PEEK(adrid+17)*2
1Ø5Ø nblig=PEEK(adrid+16)
Calcul du nombre d'octets (nboct) par lignes et définition du nombre
de lignes caractères (nblig) pour le transfert du sprite en données.
1Ø6Ø '
1Ø7Ø ' transfert des octets dessin
1Ø8Ø GOSUB 132Ø
1Ø9Ø '
11ØØ ' traitement de plusieurs dessins pour un sprite
111Ø READ n1,n2,n3
112Ø IF n1 <> Ø THEN n=n1:GOSUB 117Ø
113Ø IF n2 <> Ø THEN n=n2:GOSUB 117Ø
114Ø IF n3 <> Ø THEN n=n3:GOSUB 117Ø
115Ø GOTO 121Ø
116Ø 'sous-progr pour dessins multiples
117Ø CLS
118Ø ' sequences affichage dessins multiples
119Ø ON n GOSUB 19ØØ
12ØØ GOTO 132Ø
121Ø adrid=adrid+22
Mise à jour de l'adresse en descripteur + 22 octets.
122Ø NEXT j
Fin de la boucle de programme suivant nombre de sprites.
123Ø '------------------------
124Ø ' fin de transfert des DATA des sprites
125Ø ' enregistrer le fichier sprites
126Ø '------------------------
127Ø SAVE"sprites",b,adridØ,adrdata-adridØ

Enregistrement du fichier "sprites" en binaire, à partir de l'adresse de
début et, suivant sa longueur.
128Ø END
129Ø '*********************************
13ØØ 'sous-progr transfert octets dessin
131Ø '*********************************
132Ø '  source = adresse ecran
133Ø source=&CØØØ
L'adresse de début de la mémoire écran = &CØØØ soit en décimal
4Ø96 * 12 (C = 12) = 49152. Chaque sprite affiché dans le coin haut
gauche débute à cette adresse.
134Ø FOR li=1 TO nblig
135Ø source1=source
136Ø FOR lig=1 TO 8
Une ligne caractère contient 8 lignes graphiques.
137Ø FOR nboc=1 TO nboct
138Ø POKE adrdata,PEEK(source1)
139Ø adrdata=adrdata+1:source1=source1+1
Transfert des octets de l'écran vers le fichier des données et mise à
jour des adresses "adrdata" et "source1".
14ØØ NEXT nboc
Fin de la boucle de transfert d'une ligne graphique suivant le nombre
d'octets.
141Ø source1=source1+2Ø48-nboct
Mise à jour de l'adresse écran, la ligne graphique suivante se situe
2Ø48 octets plus loin.
142Ø NEXT lig
Fin de la boucle de transfert d'une ligne caractère (= 8 lignes graphi-
ques).
143Ø source=source+8Ø
Mise à jour de l'adresse écran plus 1 ligne caractère (intervalle entre
2 lignes de caractères + 8Ø octets). Tu vois, au passage, que l'orga-
nisation interne de la mémoire écran n'est pas spécialement évidente.
144Ø NEXT li
Fin de la boucle de transfert du sprite suivant le nombre de lignes.
145Ø RETURN
146Ø '------------------------
147Ø ' fin de GENSPRIT
148Ø '*************************
149Ø ' sous-progr affichage des sprites
15ØØ '*************************
151Ø ' ----------------------
152Ø ' afficher l'avion
153Ø ' ----------------------
154Ø '
Il s'agit de l'affichage en BASIC des caractères redéfinis, superposés
suivant le plan de l'avion, pour obtenir en mémoire écran la suite
d'octets codifiés en INK 1,2 et 3.
155Ø PRINT CHR$(22)+CHR$(1);
156Ø LOCATE x,y:PEN 2
157Ø PRINT CHR$(255);CHR$(254);
158Ø PEN 1:PRINT CHR$(253);CHR$(8);
159Ø PEN 2:PRINT CHR$(252);
16ØØ PEN 1:PRINT CHR$(251);CHR$(8);
161Ø PEN 3:PRINT CHR$(25Ø);CHR$(249);
162Ø LOCATE x,y+1
163Ø PEN 1:PRINT CHR$(248);CHR$(8);
164Ø PEN 2:PRINT CHR$(247);CHR$(8);
165Ø PEN 3:PRINT CHR$(246);
166Ø PEN 1:PRINT CHR$(245);CHR$(8);
167Ø PEN 2:PRINT CHR$(244);
168Ø PEN 1:PRINT CHR$(243);CHR$(8);
169Ø PEN 2:PRINT CHR$(242);
17ØØ PEN 1:PRINT CHR$(241);CHR$(8);
171Ø PEN 2:PRINT CHR$(24Ø);CHR$(8);
172Ø PEN 3:PRINT CHR$(239);
173Ø PEN 1:PRINT CHR$(238);CHR$(8);
174Ø PEN 2:PRINT CHR$(237);CHR$(8);
175Ø PEN 3:PRINT CHR$(236);
176Ø PEN 1:PRINT CHR$(235);CHR$(8);
177Ø PEN 2:PRINT CHR$(234);
178Ø LOCATE x+1,y+2:PRINT CHR$(233);
179Ø PEN 1:PRINT CHR$(232);CHR$(8);
18ØØ PEN 2:PRINT CHR$(231);
181Ø RETURN
182Ø '------------------------
183Ø ' adversaire caractere 227
184Ø '------------------------
185Ø LOCATE x,y:PRINT CHR$(227)
186Ø RETURN
187Ø '------------------------
188Ø ' 2eme dessin adversaire 227
189Ø '------------------------
19ØØ LOCATE x,y:PRINT CHR$(228)
191Ø RETURN
192Ø '------------------------
```

Immédiatement tu sauvegardes ton listing par: SAVE"GENSPRIT", ensuite tu fais RUN et ton fichier "sprites" existe entièrement, descripteur et données. Le prochain programme à écrire doit gérer ce fichier, une partie très courte en BASIC, une séquence beaucoup plus importante en ASSEMBLEUR. Premier objectif, afficher l'avion et l'adversaire en assurant les déplacements, l'avion commandé au joystick et l'adversaire suivant les paramètres du descripteur (inversion de la direction si les limites d'évolution sont atteintes). Chaque routine sera expliquée au pas à pas et, je te donnerai les DATA des codes machine si tu ne disposes pas de programme ASSEMBLEUR. Toutes ces routines vont constituer un kit incorporable totalement ou partiellement dans tes applications: JOYSTICK-EN-KIT. A petites doses l'ASSEMBLEUR se digère très bien, la semaine prochaine tu vas être gaté!!!

Bien joystiquement vôtre
François LE GRIGUER

LA SEMAINE PROCHAINE

La gestion du fichier "SPIRITES". Les premières routines en ASSEMBLEUR.

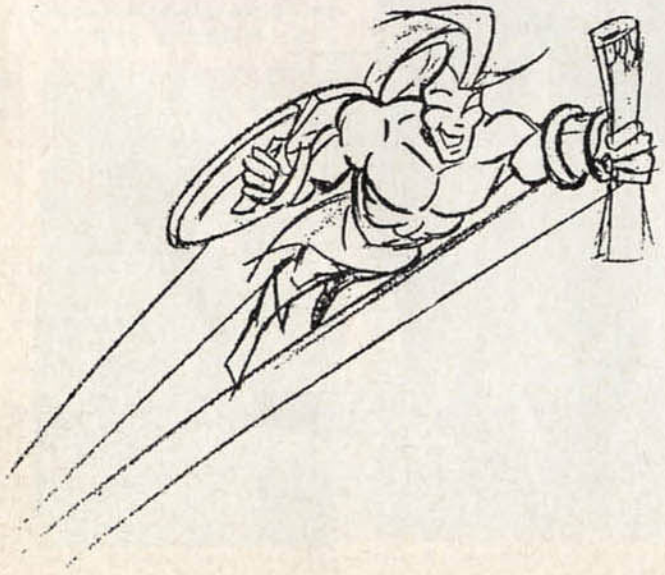
PAR FRANÇOIS LE GRIGUER

JOYSTICK EN KIT

DES SPRITES ASSEMBLES

QU'EST-CE QUE TU PENSES DE LA TRANSFORMATION D'UN SPRITE EN SPIRITE? PAS FACILE, MEME AUTOUR D'UNE TABLE RONDE A TROIS PIEDS. AUCUN RAPPORT!!! BIEN D'ACCORD AVEC TOI ET POURTANT, C'EST CE QUI S'EST PASSE LA SEMAINE DERNIERE A LA COMPOSITION DU TEXTE. TU AS DU PRENDRE CELA POUR UN GAG. DONC NOUS NE FAISONS PAS DANS LE SPIRITISME, AUJOURD'HUI NOS SPRITES S'ANIMENT EN ASSEMBLEUR, ET A TRES GRANDE VITESSE. TON AVION EVOLUE EN HAUTEUR FACE A UN ADVERSAIRE ANIME, ACCROCHE BIEN TA CEINTURE, L'ASSEMBLEUR T'ATTEND.

J'APPRENDS



Depuis la semaine dernière ton fichier "sprites" existe et, il suffit maintenant de savoir l'exploiter pour voir à l'écran les premiers déplacements. Commençons par le programme BASIC "ANISPRIT": son role se limite à charger la fichier "sprites" et la séquence en ASSEMBLEUR. Pour cette séquence deux possibilités: soit le transfert des DATA, soit le chargement du programme objet issu de l'assemblage. Ce choix est simple: tu disposes ou non d'un ASSEMBLEUR.

```
10 ****************
20 '  ANISPRIT
30 ****************
40 '
50 ' animation d'un sprite
60 ' a partir du fichier sprites
70 '----------------
80 '
90 adrid=36000:obj=32000
```

L'adresse "adrid" = adresse de début des identificateurs. L'adresse "obj" = adresse de début du programme binaire. Le programme source que tu écris en Assembleur se transforme, lors de l'assemblage, en code binaire ce qui représente le programme objet, directement exécutable par ton micro.

```
100 MEMORY obj-1:MODE 1:x=1:y=1
110 INK 0,0:INK 1,13:INK 2,9:INK 3,7
120 LOAD"sprites",adrid
```

chargement de ton fichier "spites"

```
130 ' LOAD"kitobj",obj
```

Si tu as généré ton programme objet tu utilises la ligne 130 pour son chargement.

```
131 FOR j=obj TO obj+424
```

Cette boucle de programme permet de charger le programme objet à partir des DATA qui contiennent l'ensemble des codes.

```
132 READ a$:a=VAL("&"+a$)
133 POKE j,a:NEXT j
140 " vers les routines assembleur
150 CALL obj,adrid:GOTO 150
```

Ici le programme BASIC fait appel au sous-programme Assembleur avec «CALL obj» et, communique l'adresse de début des identificateurs en ajoutant «,adrid».

```
160 " ****************************
990 " data code objet routines assembleur
```

Toutes les lignes de DATA qui suivent contiennent l'ensemble du fichier objet généré par l'assemblage du programme source. Les valeurs sont en hexadécimal (base 16). Fais très attention lors de ta saisie car la moindre erreur entrainera des anomalies de traitement.

```
1000 DATA D5,DD,E1,CD,80,7D,CD,19,BD,DD
1010 DATA 7E,01,FE,00,28,0C,CD,9A,7D,CD
1020 DATA 80,7E,CD,DD,7D,CD,29,7D,11,16
1030 DATA 00,DD,19,DD,7E,00,FE,00,20,DE
1040 DATA C9,DD,66,01,7C,FE,00,C8,DD,6E
1050 DATA 02,25,2D,CD,1A,BC,E5,DD,6E,14
1060 DATA DD,66,15,DD,46,0B,10,02,18,10
1070 DATA DD,7E,11,07,07,07,07,5F,16,00
1080 DATA DD,46,10,19,10,FD,D1,DD,46,10
1090 DATA DD,7E,11,07,32,63,7D,C5,01,00
1100 DATA 00,D5,ED,B0,D1,7A,C6,08,57,E6
1110 DATA 38,20,F1,C1,10,01,C9,E5,EB,11
1120 DATA B0,3F,ED,52,EB,E1,18,E1,CD,24
1130 DATA BB,E6,03,20,05,DD,36,04,00,C9
1140 DATA CB,44,28,05,DD,36,04,05,C9,DD
1150 DATA 36,04,01,C9,DD,7E,01,FE,00,C8
1160 DATA 3E,00,CD,90,BB,DD,66,01,DD,6E
1170 DATA 02,25,2D,CD,1A,BC,DD,46,10,DD
1180 DATA 4E,11,CB,01,79,32,BE,7D,C5,0E
1190 DATA 00,E5,06,00,5D,54,13,36,00,ED
1200 DATA B0,E1,7C,C6,08,67,E6,38,20,EB
1210 DATA C1,10,01,C9,11,B0,3F,ED,52,18
1220 DATA DF,DD,7E,04,FE,00,C8,DD,66,06
1230 DATA DD,46,07,DD,4E,08,DD,56,09,DD
1240 DATA 7E,01,B9,28,0C,BA,28,09,DD,7E
1250 DATA 02,BC,28,03,B8,20,18,DD,7E,0A
1260 DATA FE,00,20,05,DD,36,01,00,C9,DD
1270 DATA 86,04,FE,09,38,02,D6,08,DD,77
1280 DATA 04,DD,7E,05,3D,07,07,07,07,5F
1290 DATA 16,00,21,50,7E,DD,7E,04,19,3D
1300 DATA 07,83,5F,19,DD,7E,01,86,FE,26
1310 DATA 30,C9,FE,00,28,C5,DD,77,01,23
1320 DATA DD,7E,02,86,FE,10,30,B9,FE,00
1330 DATA 28,B5,DD,77,02,C9,00,FF,01,FF
1340 DATA 01,00,01,01,00,01,FF,01,FF,00
1350 DATA FF,FF,00,FE,02,FE,02,00,02,02
1360 DATA 00,02,FE,02,FE,00,FE,FE,00,FD
```

```
137Ø DATA Ø3,FD,Ø3,ØØ,Ø3,Ø3,ØØ,Ø3,FD,Ø3
138Ø DATA FD,ØØ,FD,FD,DD,7E,Ø1,FE,ØØ,C8
139Ø DATA DD,7E,12,FE,Ø1,C8,4F,DD,7E,ØB
14ØØ DATA B9,2Ø,11,DD,7E,ØC,FE,ØØ,2Ø,Ø5
141Ø DATA DD,36,Ø1,ØØ,C9,DD,36,ØB,Ø1,C9
142Ø DATA 3C,DD,77,ØB,C9
```

Tu sauvegardes ton listing avant de faire RUN, une erreur en assembleur se traduit en général par un plantage dont tu ne peux sortir que par un RESET.
Tu lances ton programme, l'avion s'affiche. Au joystick tu commandes ses déplacements verticaux et les limites sont controlées. L'adversaire évolue en diagonale, à grande vitesse, avec 2 images différentes, sa direction s'inverse dès l'atteinte d'une limite. Actuellement la vitesse d'exécution est beaucoup trop importante, les sprites ne sont pas assez nombreux et, rien ne t'empêche de créer plusieurs adversaires de la taille que tu veux, il te suffit de les incorporer dans "GENSPRIT", le générateur de sprites vu la semaine dernière.
Pourquoi et comment tout celà fonctionne? suivant le descripteur du sprite et, grace à la séquence en ASSEMBLEUR qui suit, le programme source, que l'on va prendre,point par point, accompagné d'un maximum de commentaires en italique.

L'Assembleur, un mot magique, le symbole de la vitesse et de la place minimum, mais aussi le best en programmation dont tu rêves peut-être. Réaliser un programme important en Assembleur demande beaucoup d'expérience, par contre, des routines très courtes sont à ta portée avec un minimum d'efforts.

Les régistres du microprocesseur Z8Ø:

PC (Program Counter) contient l'adresse de la prochaine instruction à exécuter
SP (Stack Pointer) gère l'adresse de la pile
AF (Accumulator/Flags) accumulateur et indicateurs
IX régistre d'Index X
IY régistre d'Index Y
A cette liste s'ajoutent six régistres: B, C, D, E, H, L.
Ces régistres 8 bits s'associent pour former les régistres 16 bits:
BC, DE et HL
qui contiennent dans l'ordre: poids fort/poids faible.
Après cette petite mise en condition on aborde le listing source.

```
1Ø ;**********************
2Ø ;    KITASS
3Ø ;**********************
4Ø ;
5Ø ;GESTION DES SPRITES CARACTERES
6Ø ;EN ENTREE (DE) = ADRID(ADRESSE DEBUT IDENTIF)
```

Tous les commentaires du programme sont précédés par le point-virgule (;) l'équivalent de l'apostrophe (') en Basic.

```
7Ø ORG 32ØØØ
```

Tout programme assembleur débute par ORG qui donne l'adresse de début pour l'assemblage (1er octet du programme objet).

```
8Ø DEB:
```

C'est un symbole adresse appelé également étiquette limité à 6 caractères maxi et défini librement par le programmeur.

```
9Ø PUSH DE
```

Cette première instruction correspond au code "D5" en première ligne DATA du programme BASIC. Le mnémonique PUSH associé au régistre DE est converti par l'assembleur en D5 (hexadécimal). Signifie empiler le contenu de DE. Ce contenu est "adrid", l'adresse ajoutée en BASIC lors du CALL obj,adrid.

```
1ØØ POP IX ;init adresse debut identificateurs
```

POP IX est converti en "DD E1", octets suivants en DATA. Ceci récupère dans la pile le contenu. PUSH DE et POP IX effectue le transfert du contenu de DE vers IX. A partir d'ici IX contient l'adresse de l'identificateur et va permettre d'accéder à n'importe lequel des 22 octets (tu reprends le plan de l'identificateur paru dans le numéro 14).

```
11Ø ;test joystick montee/descente
12Ø CALL JOY;sous-prog
```

Branchement dans le sous-programme étiquette JOY. La conversion par l'assembleur s'effectue en 2 temps. Le mnémonique est converti en "CD" et l'étiquette JOY est remplacée par son adresse dans le programme soit "8Ø 7D" (8Ø pour l'octet faible et 7D pour l'octet fort). Toutes ces valeurs figurent dans cet ordre dans les DATA du programme BASIC et, si je te les donne, c'est tout simplement pour que tu comprennes bien le lien entre ce programme source et le code objet mais, il n'y a pas à les connaitre, l'assembleur est là pour celà heureusement.

```
13Ø ;effacer le sprite
14Ø DEB1:
```

Etiquette DEB1.

```
15Ø CALL #BD19
```

Branchement dans le sous-programme système, à l'adresse BD19, pour la synchro du signal vidéo. Le symbole # veut dire hexadécimal comme & en BASIC.

```
16Ø LD A,(IX+1)
```

Chargement dans le régistre A du contenu de IX+1 c-à-d le 2ème octet de l'identificateur (valeur de x). IX pointe sur l'octet Ø. +1 est la valeur du déplacement qui s'ajoute à l'adresse contenue en IX.

```
17Ø CP Ø;sprite inactif
```

Comparaison du contenu de A avec Ø pour controler si le sprite est actif.

```
18Ø JR Z,DEB2
```

Saut en DEB2 si Z, c-à-d si le contenu de A est égal à Ø.

```
19Ø CALL EFFAC
```

Branchement dans le sous-programme EFFAC.

```
2ØØ ;maj animation du sprite
21Ø CALL ANIM
```

Branchement dans le sous-programme ANIM.

```
22Ø ;maj des coordonnees
23Ø CALL MAJ
```

Branchement dans le sous-prog MAJ.

```
24Ø ;afficher le sprite
25Ø CALL AFFIC
```

Branchement dans le sous-prog AFFIC.

```
26Ø ;suite sprite
27Ø DEB2:
28Ø LD DE,22
```

Chargement en DE de la valeur 22 (longueur d'un identificateur).

```
29Ø ADD IX,DE;progression sprite + 1
```

Addition du contenu de DE à celui de IX (adresse identificateur + 22).

```
3ØØ LD A,(IX) ;controle fin fichier
31Ø CP Ø
```

Chargement en A du 1er octet et comparaison avec Ø.

```
32Ø JR NZ,DEB1
```

Si le résultat de la comparaison est non (NZ) saut en DEB1. Sinon suite en ligne suivante.

```
33Ø RET
```

Fin de la séquence et retour dans le programme BASIC.

```
34Ø ;**********************************
```

Chaque sous-programme en assembleur sera étudié dans le détail mais je te donne maintenant la liste des mnémoniques utilisés avec leur signification.
ADD (ADDition) addition avec résultat dans le premier régistre.
ADD IX,DE = addition de IX et DE, résultat en IX.
CALL branchement dans un sous-programme (idem GOSUB en Basic).
CP (ComPare) comparaison d'une valeur ou d'un régistre avec le contenu de l'accumulateur A.
JR (Jump Relative) saut à une adresse rapprochée +127 ou -128 octets. La valeur du déplacement est tenue sur 1 octet. Un saut peut être inconditionnel, par exemple JR DEB1 effectue dans tous les cas un saut en DEB1. Un saut conditionnel est lié à une condition, par exemple JR NZ,DEB1 effectue le saut en DEB1 si le résultat du test n'est pas oui.
LD (LoaD) chargement d'une valeur ou du contenu d'un autre régistre dans le premier régistre. Exemple: LD A,B charger en A le contenu de B.
POP (dépiler) Extrait les 2 octets du dessus de la pile et les charge dans le régistre, exemple POP IX charge en IX les 2 octets du dessus de la pile.
PUSH (empiler) Place le contenu du régistre sur le dessus de la pile, exemple
PUSH DE place le contenu de DE dans la pile.
RET (RETurn) Instruction de fin de sous-programme.

Les premières routines en Assembleur fonctionnent mais doivent encore être commentées. Des compléments et de nouvelles routines seront ajoutés, dans les semaines qui viennent, pour gérer l'ensemble des codes prévus dans l'identificateur de sprite. Même si tu as un peu de mal, l'Assembleur doit te paraitre déjà beaucoup moins compliqué, et pour toutes les questions tu m'écris.

Bien Joystiquement Votre
François LE GRIGUER

LA SEMAINE PROCHAINE

Toujours de l'ASSEMBLEUR:
le test du Joystick, effacer le sprite, la mise à jour des coordonnées.

PAR FRANÇOIS LE GRIGUER

J'APPRENDS

COMMENT VONT TOUS LES ALLUMES DE LA PROGRAMMATION? TU DEVIENS UN PRO, L'ASSEMBLEUR N'AURA BIENTOT PLUS DE SECRETS POUR TOI!! ENFIN PRESQUE PLUS!! IL NOUS FAUT, ENCORE, TRAVAILLER UN PEU, ET VOIR LE DETAIL DE TOUS LES SOUS-PROGRAMMES QUI GERENT, ACTUELLEMENT, LES SPRITES. ENSUITE, CE SERONT DES COMPLEMENTS, TOUJOURS EN ASSEMBLEUR, POUR DE-CLENCHER DES ACTIONS PARTICULIERES COMME UN TIR, UNE EXPLOSION OU ENCORE L'INITIALISATION D'UN NOUVEL ADVERSAIRE, ET LE CONTROLE DES COLLISIONS.

14

ENCORE DES ROUTINES EN ASSEMBLEUR

Le programme BASIC "ANISPRIT" contient, en Data, tous les codes des routines Assembleur. Même si tu ne disposes pas d'un Assembleur tu dois les connaitre et les comprendre. Aujoud'hui, on regarde ensemble le test du Joystick, l'effacement et la mise à jour des coordonnées des sprites, des sous-programmes qui sont appelés par la séquence principale de la semaine dernière. Tous les textes en italique sont des explications et, tu dois te référer à l'identificateur de sprite, publié dans le numéro 14, pour bien suivre le traitement.

```
970 ;*****************************
980 ;sous-programme test joystick
990 ;*****************************
1000 JOY:
1010 CALL #BB24
```

Appel de la routine système qui donne en sortie l'état du Joystick, dans les régistres A et H pour le Joystick Ø, en L pour le joystick 1 (2 Joysticks possibles). Chaque bit, s'il est à 1, indique la position du Joystick. Bit Ø = haut. Bit 1 = bas. Bit 2 = gauche. Bit 3 = droite. Bit 4 = fire2. Bit 5 = fire1.

```
1020 AND 3
```

Conserver en A la valeur des bits Ø et 1.

```
1030 JR NZ,JOY1
```

Si le contenu est différent de Ø suite en JOY1.

```
1040 LD (IX+4),Ø;Ø en direction
```

Pas d'ordre Joystick mettre Ø en direction du sprite et fin du sous-programme par RET.

```
1050 RET
1060 JOY1:
1070 BIT Ø,H
```

Controle de la valeur du bit Ø (Joystick en haut).

```
1080 JR Z,JOY2
```

Si bit Ø = Ø suite en JOY2.

```
1090 LD (IX+4),5 ;direction 5
```

Bit Ø = 1 = joystick en haut = descente de l'avion = direction 5 et fin du sous-programme par RET.

```
1100 RET
1110 JOY2:
1120 LD (IX+4),1;direction 1
```

Si la valeur des bits Ø et 1 est différente de zéro et que le bit Ø = Ø donc le bit 1 = 1 = joystick en bas = montée de l'avion = direction 1. Fin du sous-prog par RET.

```
1130 RET
```

Le test du Joystick controle la montée et la descente. Par la suite tu y ajouteras le test des FIRE. Il est plus simple de marcher par étape.

```
1140 ;*****************************
1150 ;sous-prog effacer sprite
1160 ;*****************************
```

La première phase, dans le déplacement d'un sprite, c'est de l'effacer avant de mettre à jour ses coordonnées. Toujours pour rester simple, l'effacement ne restaure pas les décors. Tu considères que les sprites évoluent sur un fond neutre en INK Ø. La sauvegarde des décors se fera par la suite.

```
1170 EFFAC:
1180 LD A,(IX+1);si x=Ø sprite inactif
1190 CP Ø
1200 RET Z
```

Controle de la valeur x = Ø. Si oui le sprite est inactif et fin du sous-prog.

```
1210 LD A,Ø
1220 CALL #BB9Ø;pen Ø
```

Charger en A la valeur Ø (INK Ø) et appel de la routine système qui initialise PEN.

```
1230 LD H,(IX+1);valeur de x
1240 LD L,(IX+2);valeur de y
```

Charger en H et en L les valeurs x et y du sprite.

```
1250 DEC H
1260 DEC L
```

Soustraire 1 au contenu des régistres H et L pour l'utilisation de la routine système.

```
1270 CALL #BC1A ;ADRESSE ECRAN
```

Cette routine donne en sortie l'adresse écran en HL.

```
1280 LD B,(IX+16) ;nombre de lignes caract
1290 LD C,(IX+17) ;nombre de caracteres
```

Chargement du nombre de lignes et du nombre de caractères à partir des données de l'identificateur de sprite. Il s'agit de données en mode caractère.

```
1300 RLC C ;nb caract*2 = nb octets
```

Multiplication du nombre de caractères par 2 (1 caractère = 2 octets), par rotation à gauche d'un bit (si le bit Ø = 1, décalé d'un bit à gauche c'est maintenant le bit 1 = 1 et la valeur 1 devient 2).

```
1310 LD A,C
1320 LD (NBOCT),A
```

Les ordres LD A,C et LD (NBOCT),A sont nécessaires pour transférer le nombre d'octets en NBOCT. Pourquoi pas directement LD (NBOCT),C, parce que cette instruction n'existe pas.

```
1330 EFFAC1:
1340 PUSH BC
```

Sauvegarde des régistres B et C dans la pile.

```
1350 EFFAC2:
1360 LD C,Ø
```

Charger en C la valeur directe du nombre d'octets par ligne.

```
1370 NBOCT:EQU $-1
```

Déclaration de l'adresse NBOCT.

```
1380 PUSH HL
```

Sauver dans la pile l'adresse écran contenue dans HL.

```
1390 LD B,Ø
```

```
14ØØ LD E,L
141Ø LD D,H
142Ø INC DE
143Ø LD (HL),Ø
144Ø LDIR
```
L'instruction LDIR est très performante: transfert de l'emplacement mémoire adressé par HL vers l'emplacement mémoire adressé par DE, incrémentation de DE et de HL (adresse suivante) et décrémentation de BC. Si BC est différent de Ø le transfert est renouvelé. Ici pour mettre Ø dans tous les octets de la ligne du sprite, tu transféres en DE le contenu de HL (adresse écran), tu incrémentes DE (adresse+1), tu mets Ø en adresse (LD (HL),Ø) et Ø en B (LD B,Ø), C contient le nombre d'octets.

```
145Ø ;nouvelle ligne graphique
146Ø POP HL ;adresse ecran debut de ligne
147Ø LD A,H
148Ø ADD A,8;+ 2Ø48
149Ø LD H,A
```
Pour passer à la ligne graphique suivante: récupérer en HL l'adresse écran précédente et ajouter 2Ø48 (intervalle entre 2 lignes graphiques à l'intérieur d'une ligne de caractères). Additionner 2Ø48 à HL revient à ajouter 8 à H, l'octet fort de l'adresse (voir la valeur des bits dans un régistre 16 bits).

```
15ØØ AND 56;controle 8 lignes
```
Cette instruction donne en A la valeur des bits 3, 4, 5 de l'octet fort. Si cette valeur = Ø celà signifie que les 8 lignes graphiques d'un caractère sont faites, la nouvelle adresse sort de la mémoire écran.

```
151Ø JR NZ,EFFAC2
```
Si la valeur est # de Ø retour en EFFAC2.

```
152Ø POP BC;nb de lignes
```
Une ligne de caractères vient d'être réalisée. BC contient le nombre de lignes à effacer.

```
153Ø DJNZ EFFAC3
```
Encore une instruction puissante avec DJNZ: décrémenter B (-1), si B # Ø suite en EFFAC3, sinon suite à la ligne suivante.

```
154Ø RET
```
Fin du sous-programme.

```
155Ø EFFAC3:
156Ø LD DE,163Ø4
157Ø SBC HL,DE
```
Début d'une nouvelle ligne de caractères, l'adresse écran est mise à jour par soustraction de 163Ø4 (16384 - 8Ø), 8Ø étant l'intervalle entre 2 lignes de caractères.

```
158Ø JR EFFAC1 ;retour nouvelle ligne
159Ø ;**********************************
16ØØ ;sous-prog maj des coordonnees
161Ø ;**********************************
162Ø MAJ:
163Ø LD A,(IX+4);direction
164Ø CP Ø
165Ø RET Z
```
Première démarche: controler si la direction est # de Ø. Si oui fin du sous-prog, pas de mise à jour des coordonnées.

```
166Ø ;test limites atteintes
167Ø LD H,(IX+6);HAUT
168Ø LD B,(IX+7);BAS
169Ø LD C,(IX+8);GAUCHE
17ØØ LD D,(IX+9);DROITE
```
Chargement en H, B, C et D des limites d'évolution prévues dans le descripteur du sprite.

```
171Ø LD A,(IX+1);X
172Ø CP C
173Ø JR Z,MAJ1
174Ø CP D
175Ø JR Z,MAJ1
```
Controle de la valeur X avec les limites gauche et droite.

```
176Ø LD A,(IX+2);Y
177Ø CP H
178Ø JR Z,MAJ1
179Ø CP B
18ØØ JR NZ,MAJ2
```
Controle de la valeur Y avec les limites haut et bas.

```
181Ø ;limite atteinte
182Ø MAJ1:
183Ø LD A,(IX+1Ø);code limite atteinte
184Ø CP Ø
185Ø JR NZ,MAJ11
```
Si une limite est atteinte le code en descripteur indique la suite du sprite: si = Ø à supprimer, sinon inverser la direction en cours.

```
186Ø ;sprite elimine
187Ø LD (IX+1),Ø
188Ø RET
```
Pour éliminer un sprite mettre Ø en valeur X.

```
189Ø MAJ11:
19ØØ ;inverser direction
191Ø ADD A,(IX+4)
```
Le régistre A contient 4 à additionner à la direction pour l'inverser.

```
192Ø CP 9
193Ø JR C,MAJ12
194Ø SUB 8
```
La nouvelle direction doit être inférieure à 9, sinon soustraire 8.

```
195Ø MAJ12:
196Ø LD (IX+4),A
```
Transfert en identificateur de la nouvelle direction.

```
197Ø ;mise a jour x et y
198Ø MAJ2:
```
La mise à jour s'effectue par l'intermédiaire de la table TABXY qui contient, pour 3 vitesses, la valeur + ou - en x et en y par direction. Chaque vitesse représente donc 16 octets (val de x et val de y = 2 octets * 8 directions).

```
199Ø LD A,(IX+5);vitesse
2ØØØ DEC A;-1
```
Charger en A la vitesse et soustraire 1 pour le début du fichier.

```
2Ø1Ø RLCA
2Ø2Ø RLCA
2Ø3Ø RLCA
2Ø4Ø RLCA;multiplier par 16
```
La multiplication par 16 s'obtient par 4 rotations à gauche des bits.

```
2Ø5Ø LD E,A
2Ø6Ø LD D,Ø
2Ø7Ø LD HL,TABXY
2Ø8Ø LD A,(IX+4);direction
2Ø9Ø ADD HL,DE
```
HL contient l'adresse de début de la table et DE la valeur du déplacement en fonction de la vitesse (+Ø ou +16 ou +32 octets). Avec ADD HL,DE le régistre HL pointe dans la table sur la direction 1 de la vitesse recherchée.

```
21ØØ DEC A;-1
211Ø RLCA;multiplie par 2
212Ø ADD A,E
213Ø LD E,A
```
La direction en A décrémenté et multiplié par 2 (2 octets par direction) transféré en E.

```
214Ø ADD HL,DE;adresse en table +- x
```
Après cette nouvelle addition HL pointe sur la valeur de mise à jour de X.

```
215Ø LD A,(IX+1)
216Ø ADD A,(HL)
217Ø CP 38;controle des limites ecran
```
Addition de X en cours avec la valeur de mise à jour et controle limites écran.

```
218Ø JR NC,MAJ1
```
Si la limite est atteinte retour en MAJ1.

```
219Ø CP Ø
22ØØ JR Z,MAJ1
```
Idem pour limite gauche.

```
221Ø LD (IX+1),A;nouvelle valeur X
```
Mise à jour de la nouvelle valeur de X en identificateur.

```
222Ø INC HL;pour +- y
223Ø LD A,(IX+2)
224Ø ADD A,(HL)
225Ø CP 16;controle des limites
226Ø JR NC,MAJ1
227Ø CP Ø
228Ø JR Z,MAJ1
229Ø LD (IX+2),A;nouvelle valeur Y
```
Même processus pour la mise à jour de Y.

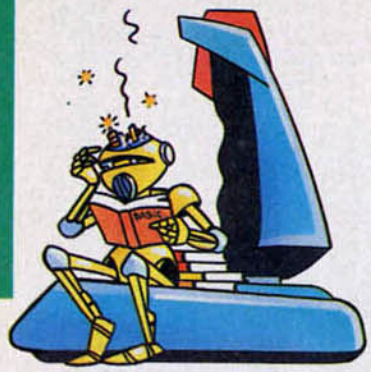
```
23ØØ RET
231Ø ;**************************
232Ø ;TABLE DES VALEURS +- X ET Y PAR VITESSE
233Ø ;3 VITESSES PREVUES 1 2 3
234Ø ;VALEUR X Y PAR 8 DIRECTIONS POUR CHAQUE VITESSE
235Ø ;
236Ø TABXY:
237Ø ;vitesse=1
238Ø DEFB Ø,255,1,255,1,Ø,1,1,Ø,1,255,1,255,Ø,255,255
239Ø ;vitesse 2
24ØØ DEFB Ø,254,2,254,2,Ø,2,2,Ø,2,254,2,254,Ø,254,254
241Ø ;vitesse 3
242Ø DEFB Ø,253,3,253,3,Ø,3,3,Ø,3,253,3,253,Ø,253,253
243Ø ;**************************
```

Tu t'en es sorti. Super!! Pas simple, ça ne s'improvise pas l'Assembleur, ne t'affole pas je t'explique au pas à pas, dans les semaines qui viennent tu comprendras encore mieux, et puis on fera peut-être une pause dans l'Assembleur pour revenir à des programmes Basic. Si tu m'écrivais pour me dire ce que tu en penses, bonne idée non!!

Bien Joystiquement Votre
François LE GRIGUER

LA SEMAINE PROCHAINE

Les Routines ASSEMBLEUR: mise à jour de l'animation et affichage du sprite.



PAR FRANÇOIS LE GRIGUER

JOYSTICK EN KIT

DEPUIS LE NOTRE PREMIER NUMERO L'AMSTRAD EST PRIVILEGIE DANS J'APPRENDS AVEC DES PROGRAMMES EN BASIC ET EN ASSEMBLEUR MAIS IL EST TEMPS DE PENSER A TOUS LES AUTRES ET VOUS ETES TRES NOMBREUX, A ATTENDRE UNE INITIATION POUR VOTRE MICRO PREFERE. JE VOUS EN DIRAI PLUS LA SEMAINE PROCHAINE. AUJOURD'HUI ON CONTINUE A ETUDIER TOUS LES DETAILS DES ROUTINES ASSEMBLEUR Z8Ø POUR LA GESTION DES SPRITES.

J'APPRENDS

ANIMATION ET AFFICHAGE EN ASSEMBLEUR

Avant de plonger, sans bouée, dans les routines on voit rapidement la pile. La pile est une zone mémoire, en RAM, d'une importance capitale dans les programmes en Assembleur. Elle suit le principe LIFO, c-à-d, Last In First Out, ce qui veut dire que la dernière donnée entrée sortira la première. La pile reçoit les données transmises par le programme (PUSH empiler et POP dépiler), mais, également les adresses de retour pour tous les accès aux sous-programmes et la gestion des interruptions. La pile est gérée par mot de 16 bits (2 octets) et l'adresse en cours se trouve dans le régistre SP. Toute erreur de programme dans l'utilisation de la pile entraine un plantage garanti comme, par exemple, une adresse de retour remplacée par une donnée.
Si tu n'as pas d'assembleur, tous les codes objet des routines font partie de "ANISPRIT" publié dans le numéro 16, et pour bien comprendre le détail des opérations tu dois, absolument, te reporter à l'identificateur de sprite du numéro 14.

```
243Ø ;*****************************
244Ø ;sous-prog maj animation du sprite
245Ø ;*****************************
```
Ce sous-programme assure la mise à jour du numéro de dessin pour les sprites qui en comportent plusieurs.
```
246Ø ANIM:
247Ø LD A,(IX+1)
248Ø CP Ø
249Ø RET Z;inactif
```
Pas de maj si le sprite est inactif (x=Ø).
```
25ØØ LD A,(IX+18);nb dessins
```
Le nombre de dessins se trouve dans l'identificateur en position 18.
```
251Ø CP 1
252Ø RET Z;un seul dessin
```
Si nombre de dessins = 1 pas de maj.
```
253Ø LD C,A
254Ø LD A,(IX+11);num dessin en cours
```
Le numéro de dessin en cours est en position 11 de l'identificateur.
```
255Ø CP C
256Ø JR NZ,ANIM1
```
Si le numéro en cours n'est pas égal au nombre de dessins suite en ANIM1.
```
257Ø LD A,(IX+12);CODE FIN ANIM
```
En fin d'animation (num. en cours = nb. dessins) controle du code suite en position 12 de l'identificateur.
```
258Ø CP Ø
259Ø JR NZ,ANIM2
26ØØ LD (IX+1),Ø;supprimer le sprite
261Ø RET
```
Le code suite = Ø donc suppression du sprite (Ø en x).
```
262Ø ANIM2:
263Ø LD (IX+11),1;retour au premier dessin
264Ø RET
```
Le code suite # de Ø donc retour du numéro en cours à 1.
```
265Ø ANIM1:
266Ø INC A
267Ø LD (IX+11),A ;+1 numero de dessin
268Ø RET
```
Le numéro en cours est incrémenté et maj en identificateur. Fin du sous-programme.
```
*************************************

34Ø ;sous-progr affichage sprite
35Ø ;*********************************
```
Un sous-programme qui affiche le sprite en recopiant la zone octets des données vers la mémoire écran, à l'adresse x, y du sprite, sans tenir compte d'un décor de fond. La gestion d'un fond sera pour une étape plus poussée.
```
36Ø AFFIC:
37Ø LD H,(IX+1);X
38Ø LD A,H
39Ø CP Ø
4ØØ RET Z;sprite inactif
```
Toujours le controle du sprite inactif.
```
41Ø LD L,(IX+2);Y
42Ø ;calcul adresse ecran
```
Ce calcul de l'adresse écran a déjà été vu dans le sous-programme d'effacement et reste identique.
```
43Ø DEC H
44Ø DEC L
45Ø CALL #BC1A
46Ø PUSH HL ;save adresse ecran
```
Les 16 bits de HL sont empilés (transfert dans la pile).
```
47Ø LD L,(IX+2Ø) ;ADRESSE BASSE
```

```
48Ø LD H,(IX+21);ADRESSE HAUTE
```
Transfert en HL de l'adresse des données (octets codifiés) à partir des positions 2Ø et 21 de l'identificateur en tenant compte des parties basse et haute.
```
49Ø LD B,(IX+11) ;numero dessin en cours
```
Il s'agit maintenant de définir l'adresse de début des données si le numéro en cours est différent de 1.
```
5ØØ DJNZ AFFIC1
```
L'instruction DJNZ décrémente B et suite en AFFIC1 si B # Ø.
```
51Ø JR AFFIC3
```
Suite en AFFIC3 si B = Ø après DJNZ.
```
52Ø AFFIC1:
53Ø ;calcul adresse en donnees
```
Le nombre de caractères multiplié par 16 donne le nombre d'octets d'une ligne de caractères (1 caractère = 8 lignes graphiques * 2 octets par ligne). Ce nombre est ensuite multiplié par le nombre de lignes caractère.
```
54Ø LD A,(IX+17) ;nb caracteres
```
Nombre de caractères en position 17 de l'identificateur.
```
55Ø RLCA
56Ø RLCA
57Ø RLCA
58Ø RLCA ;* 16 = NB octets
59Ø LD E,A
6ØØ LD D,Ø
61Ø LD B,(IX+16) ; nb lignes
62Ø AFFIC2:
63Ø ADD HL,DE
64Ø DJNZ AFFIC2
```
Boucle de programme qui répète l'addition du nombre d'octets par ligne en DE à l'adresse des données en HL.
```
65Ø AFFIC3:
66Ø POP DE;adresse ecran en DE
```
Transfert en DE à partir de la pile de l'adresse écran.
```
67Ø LD B,(IX+16) ;NB LIGNES
```
Le registre B contient le nombre de lignes caractère à afficher.
```
68Ø LD A,(IX+17) ;nb caracteres
69Ø RLCA
7ØØ LD (NBOC),A
```
Le nombre de caractères multiplié par 2 donne le nombre d'octets et transfert du nombre dans l'emplacement mémoire NBOC.
```
71Ø LIGNE:
72Ø PUSH BC
```
Sauver le contenu de BC en pile.
```
73Ø LIGNE1:
74Ø LD BC,ØØ
75Ø NBOC:EQU $-2
```
Déclaration de l'adresse de NBOC, adresse en cours - 2 octets, donc dans le premier octet qui suit l'instruction LD BC. La valeur de NBOC va se charger en C de BC.
```
76Ø PUSH DE
```
Sauver l'adresse écran de début de ligne.
```
77Ø LDIR
```
L'instruction LDIR, comme on l'a déjà vu, transfère dans l'octet adressé par DE le contenu de l'octet adressé par HL. Ensuite HL et DE sont incrémentés et BC décrémenté. L'opération se renouvelle tant que BC # Ø. En clair les octets des données sont transférés en écran pour une ligne graphique.
```
78Ø ;nouvelle ligne graphique
79Ø POP DE;adresse ecran debut ligne
```
Reprendre en DE l'adresse écran de début de ligne.
```
8ØØ LD A,D
81Ø ADD A,8; + 2Ø48
82Ø LD D,A
```
Ajouter 2Ø48 soit 8 en octet fort pour la prochaine ligne graphique.
```
83Ø AND 56
84Ø JR NZ,LIGNE1
```
Controler si la ligne de caractère (8 lignes graphiques) est terminée.
```
85Ø POP BC
```
Reprendre en BC le nombre de lignes de caractères à réaliser.
```
86Ø DJNZ LIGNE2
```
Si B est différent de Ø suite en LIGNE2, sinon fin du sous-programme.
```
87Ø RET
88Ø LIGNE2:
89Ø PUSH HL
```
Sauver le contenu de HL (adresse en données).
```
9ØØ EX DE,HL
```
Echange du contenu des régistres DE et HL. HL contient maintenant l'adresse écran.
```
91Ø LD DE,163Ø4
92Ø SBC HL,DE
```
Charger en DE 163Ø4 et soustraction de la valeur de HL pour l'adresse du début de la nouvelle ligne de caractère.
```
93Ø EX DE,HL
```
Echange de DE avec HL. DE contient à nouveau l'adresse écran.
```
94Ø POP HL
```
Reprendre en HL l'adresse des données.
```
95Ø JR LIGNE
```
Retour en LIGNE pour le transfert d'une nouvelle série de 8 lignes graphiques.
```
96Ø ;*****************************
```

L'Assembleur, un langage très simple et assez compliqué à la fois. Simple par la présentation des lignes avec une seule opération et compliqué par la multiplication de ces mêmes lignes. Tout est décomposé en ordres élémentaires, rien ne doit être omis, pas le plus petit transfert. Pour bien programmer en assembleur tu dois parfaitement bien maitriser le Basic. Passer du Basic à l'assembleur nécessite un gros travail, avec un processus de raisonnement assez différent. Ensuite, pour aborder un nouvel assembleur, il s'agit seulement de bien assimiler les nouveaux mnémoniques et connaitre les contraintes du microprocesseur.

Tu possèdes actuellement les ingrédients de base pour animer plusieurs sprites, les créer dans le programme GENSPRIT (numéro 15), les initialiser par un complément dans le programme ANISPRIT (numéro 16) en fonction de critères de temps par exemple. Que peut-on ajouter? le déclenchement de tirs et le controle des collisions avec la génération d'explosions. Refléchis bien à ces fonctions supplémentaires.

Pour terminer un petit message destiné à Laurent FISSEUX et à quelques autres qui rencontrent des difficultés dans JOYSTICK-BREAKER et la gestion de la balle. Il faut revoir l'initialisation du tableau et, en particulier, les codes du cadre dans les lignes 435Ø à 439Ø, et ça repart fort!!!

Bien Joystiquement Votre
François LE GRIGUER

LA SEMAINE PROCHAINE

UNE GRANDE NOUVEAUTE DANS J'APPRENDS, A NE MANQUER SOUS AUCUN PRETEXTE !!!