programmez!
#192 - Janvier 2016
le magazine du développeur

Drupal™ 8

DOSSIER SPÉCIAL 14 pages

Les réalités alternatives

Oculus, Hololens, Google Glass...

*Mon 1er projet **Oculus** et **Unity 5.1***
Les usages, les matériels.

SPECIAL VINTAGE !

Amiga, Amstrad CPC,
Apple II, Atari ST, GameBoy

Redécouvrez ces machines mythiques et comment les programmer

Créer son drone de A à Z

1ere partie

Connecter des objets avec **Node.JS**

Les profils en **Java 8**

M 04319 - 192 - F: 5,95 € - RD

Printed in EU - Imprimé en UE - BELGIQUE 6,45 € SUISSE 12 FS - LUXEMBOURG 6,45 € - DOM Surf 6,90 € Canada 8,95 $ CAN - TOM 940 XPF - MAROC 50 DH

L'Amstrad CPC

Dans les années 80, Amstrad n'est encore qu'une firme britannique spécialisée dans le matériel HIFI et vidéo dont le fondateur, Alan Michael Sugar (AMStrad) ambitionne d'attaquer le marché de la micro -informatique.

Baptiste Bideaux
Consultant en Digital Learning
chez PwC France.

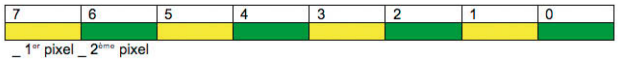
Le projet « Colour Personal Computer » démarre au début de l'année 1983, et en 1984 apparaît sur le marché l'Amstrad CPC 464, ordinateur 8bits avec 64ko de RAM, équipé d'un microprocesseur Zilog Z80 cadencé à 4MHz, d'un magnétophone à cassette et d'un magnifique clavier QWERTY coloré. Il faudra attendre 1 an pour voir arriver la version disquette de la machine : le CPC 664. La même année débarque le CPC 6128, une version du 664 avec une RAM augmentée à 128 ko. La mémoire primaire (64 ko de RAM) est partagée en 4 banques de 16 ko chacune (numérotées de 0 à 3). Le CPC 6128 propose 64ko supplémentaires qui sont eux aussi partagés en 4 banques de 16 ko accessibles en assembleur.

Affichage

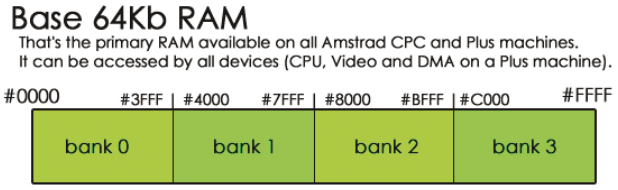
Equipé d'un moniteur monochrome ou couleur (selon les finances de chacun), la gamme CPC propose 3 modes d'affichage :
MODE 0 : 160x200 en 16 couleurs
MODE 1 : 320x200 en 4 couleurs
MODE 2 : 640x200 monochrome
La banque 3 située entre l'adresse &C000 et &FFFF est réservée pour l'affichage écran. Quel que soit le mode d'affichage, il y a toujours 200 lignes. Chaque ligne à une taille de 80 octets. Ainsi la première ligne démarre à &C000 et se termine à &C049. Chaque ligne est dessinée de manière linéaire, de gauche à droite. Cependant, à la fin d'une ligne le tracé effectue un saut de 8 pixels plus bas. Si &C000 correspond aux coordonnées (0,0), &C050 correspond à (0,8). Pour accéder à la ligne directement en dessous d'une autre, il faut incrémenter 1000 (&800) à l'adresse de cette dernière. Donc la seconde ligne sur notre écran (0,1) se trouve à l'adresse &C800.
La synchronisation vidéo est gérée par le CRTC (Cathode Ray Tube Controller), mais les couleurs sont gérées par le Gate Array qui ne raisonne pas en binaire, mais en état (0,1 ou 2). C'est pour cela que nous avons une palette de 27 teintes et non 16 ou 32. Les couleurs utilisables (16 maximum en MODE 0) peuvent être choisies parmi cette palette de teintes. Attention, les teintes de palette sont indexées différemment selon que l'on y accède par le BASIC ou par le Gate Array.

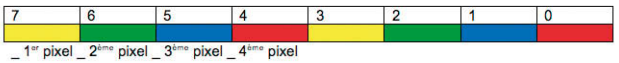
Organisation des pixels

Selon le MODE d'affichage choisi, la manière de coder les pixels est sensiblement différente.
En MODE 0, un octet suffit pour afficher 2 pixels à l'écran, un pixel ayant une longueur de 4 bits. Ainsi, la couleur du premier pixel est codée sur les bits 7, 5, 3 et 1. Le second pixel sur les bits 6, 4, 2, 0. C'est ce que l'on appelle l'entrelacement des pixels.

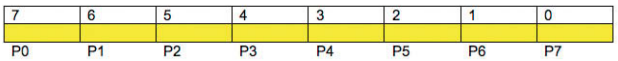


En MODE 1, l'entrelacement est identique au MODE 0. Cependant, le pixel n'est plus codé sur 4 bits, mais sur 2.





En MODE 2, c'est plus simple. Chaque bit d'un octet correspond à 1 pixel.



Comme on peut le voir, le 1er pixel se situe au bit N° 7.

La programmation

Toute la gamme CPC est livrée avec un BASIC intégré dans la ROM (le Locomotive BASIC), qui permet à l'utilisateur de faire ses premiers programmes assez rapidement. Le langage est vraiment très complet pour l'époque et fournit beaucoup de commandes pour dessiner sur l'écran, jouer des sons, gérer les périphériques (comme les 2 manettes de jeu). A noter que le BASIC du CPC 6128 propose des commandes qui n'existent pas dans le BASIC du 464 et 664. A l'époque, certains programmes étaient donc uniquement prévus pour le 6128. Le BASIC reste interprété. Aucun compilateur n'existe à l'époque. On peut toutefois protéger son code contre son listage (afin de garder un droit de propriété) et son interruption d'exécution. Même s'il reste plus lent qu'un programme codé en assembleur, le BASIC de l'Amstrad permet tout de même de produire des programmes de jeux honorables. Les puristes vous diront que pour développer pour l'Amstrad CPC il faut un Amstrad CPC. Mais bon, ceux qui n'ont pas la chance de posséder un vrai CPC peuvent se tourner vers les émulateurs qui feront très bien l'affaire. Le plus utilisé est sans aucun doute WinAPE (http://www.winape.net), qui a l'avantage d'émuler tous les Amstrad CPC et de proposer un assembleur pour le Zilog Z80. Il émule par défaut la gamme CPC+, mais le BASIC reste identique aux versions CPC plus anciennes. Téléchargez l'archive ZIP depuis le lien http://www.winape.net/download/WinAPE20A18.zip et extrayez tous les fichiers dans un dossier (« winape » par exemple). Exécutez le fichier WinAPE.exe. L'émulateur se lance et vous voilà sur l'écran d'accueil du BASIC.

Jouons avec les modes

Par défaut, le mode d'affichage est en MODE 1 (320x200 en 4 couleurs), soit 40 colonnes sur 25 lignes de caractères.
Saisissez le code suivant, en appuyez sur ENTREE pour l'exécuter :

```
MODE 0 ↵
```

L'affiche change. L'écran est nettoyé et la résolution est passée en 160x200 en 16 couleurs, soit 20 colonnes sur 25 lignes de caractères. Les caractères sont désormais 2 fois plus larges qu'en MODE 1.
Saisissez le code suivant, en appuyez sur ENTREE pour l'exécuter :

```
MODE 2↵
```

L'affiche change à nouveau et la résolution est passée en 640x200 en monochrome, soit 80 colonnes sur 25 lignes de caractères. Les caractères sont désormais 2 fois moins larges qu'en MODE 1.
Saisissez le code suivant, en appuyez sur ENTREE pour l'exécuter :

```
MODE 1↵
```

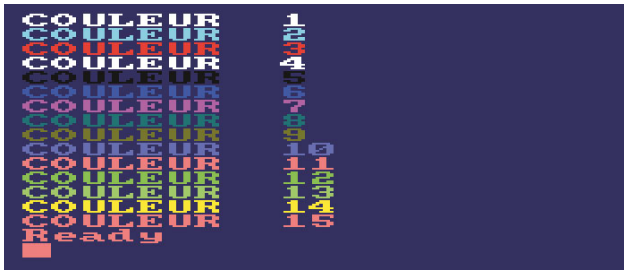
L'affichage est revenu à son état initial.

Un peu de couleurs

Le BASIC utilise les couleurs indexées de 0 à 15. Chacune est associée par défaut à l'une des 27 teintes de la palette du CPC. Saisissez le code suivant :

```
10 MODE 0↵
20 FOR R=0 TO 15↵
30 PEN R :PRINT « COULEUR » ;STR$(R)↵
40 NEXT R↵
RUN↵
```

Nous venons d'écrire un programme qui sera exécuté à votre demande. Pour exécuter un programme BASIC, vous devez saisir RUN et appuyer sur la touche ENTREE.



Voici les 16 couleurs par défaut que vous pouvez utiliser en BASIC. La couleur 0 est la couleur utilisée pour le fond d'écran (bleu nuit). Les couleurs 14 et 15 sont clignotantes car 2 teintes leur ont été associées. En effet, le BASIC permet de définir 2 teintes pour une couleur afin de créer un effet de clignotement.

Explication du programme :
Ligne 10 : Passage de l'affichage en MODE 0
Ligne 20 : Début de la boucle
Ligne 30 : On utilise PEN pour la sélection de la couleur du texte et PRINT pour afficher le texte.
Ligne 40 : Fin de la boucle
Restez sur cet écran, et saisissez ceci :

```
INK 1,15↵
```

Après exécution, la couleur 1 a changé et est devenue Orange. La commande INK permet d'associer une teinte de la palette à un index de couleur. La teinte doit être comprise entre 0 et 26. Maintenant, saisissez ceci :

```
INK 2, 5 ,4↵
```

La couleur 2 est devenue clignotante car nous avons associé 2 teintes à cette couleur (5 et 4). Que ce passe-t-il si nous changeons de mode d'affichage ?

```
10 MODE 1↵
RUN↵
```

Exécutez à nouveau le programme en saisissant RUN et en appuyant sur la touche ENTREE.

Voilà comment s'affichent nos couleurs en MODE 1. Nous avons bien nos 16 couleurs, mais seulement 4 teintes différentes (les teintes associées aux couleurs 0, 1, 2 et 3). En effet, le MODE 1 ne permet pas d'afficher plus de teintes. Essayons avec le MODE 2, saisissez le code suivant et exécuter le programme :

```
10 MODE 2↵
RUN↵
```

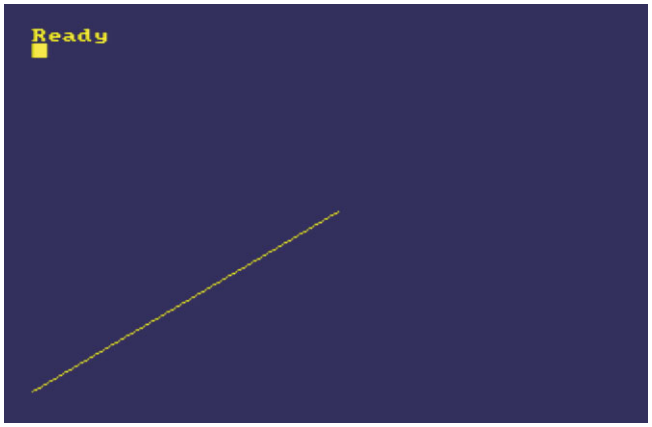
En MODE 2, seules 2 teintes sont affichables (les teintes associées aux couleurs 0 et 1).

Dessinons un peu

L'écran du CPC est orthonormé et a pour origine le coin inférieur gauche. L'axe des abscisses (X) évolue de 0 à 639, l'axe des ordonnées (Y), de 0 à 399. Et cela, quel que soit le MODE d'affichage utilisé. Il faudra donc bien avoir en tête l'organisation de ce plan avant de produire des tracés sur l'écran. Dès le démarrage du CPC, le curseur graphique est placé en coordonnées absolues (0,0). Ce curseur se déplacera en fonction de nos tracés et restera positionné sur le dernier point donné.
Commençons par supprimer le programme précédent :

```
NEW↵
10 MODE 1↵
20 MOVE 0,0↵
30 DRAW 319,199↵
RUN↵
```

Ce programme trace à l'écran une ligne partant du point(0,0) et s'arrêtant au point(319,199). Comme vous pouvez le constater, la ligne part du point inférieur gauche et s'arrête au centre de l'écran. Comme indiqué plus haut, l'axe des abscisses (X) évolue de 0 à 639, l'axe des ordonnées (Y), de 0 à 399. Le point (319,199) correspond donc au centre de l'écran.



Explication du programme :
Ligne 10 : Affichage en MODE 1
Ligne 20 : Place le curseur graphique aux coordonnées absolues (0,0)
Ligne 30 : Trace une ligne depuis la position du curseur graphique jusqu'au point (319,199)

Saisissez désormais ce code :

```
30 DRAW 639,399
RUN
```

La ligne traverse l'écran. En modifiant le MODE d'affichage (ligne 10 du programme), vous constaterez que le tracé reste identique. Seuls les pixels composant le tracé seront différents (plus larges en MODE 0, plus fins en mode 2).

Redéfinir les caractères de l'Amstrad.

L'une des possibilités offerte par le BASIC Amstrad est de pouvoir redessiner la table de caractères du CPC. Saisissez ce programme :

```
NEW
10 MODE 1
20 SYMBOL AFTER 47
30 SYMBOL 48,254,130,130,130,226,226,254,0
RUN
```

Il ne semble rien se produire. Cependant si vous saisissez le texte « 007 », vous constaterez que la forme du caractère « 0 » a changé.

Explication du programme :

Ligne 10 : Affiche en MODE 1

Ligne 20 : SYMBOL AFTER indique que la redéfinition des caractères démarre après l'index 47.

Ligne 30 : SYMBOL définit le caractère 48 (code ASCII du chiffre « 0 »).

Un caractère CPC est contenu dans une matrice 8 x 8. Chaque ligne de la matrice est codée sur 8 bits, le bit 7 correspondant au premier pixel de la ligne. Chaque pixel d'une ligne correspond à une valeur.

128	64	32	16	8	4	2	1	
■	■	■	■	■	■	■		254
■						■		130
■						■		130
■						■		130
■	■	■				■		226
■	■	■				■		226
■	■	■	■	■	■	■		254
								0

Note : *Vous pouvez modifier les caractères de 32 à 255. Les caractères 0 à 31 sont appelés caractères de contrôle et ne peuvent pas être redéfinis par l'utilisateur. Attention, la redéfinition des caractères 32 à 128 aurat un impact direct sur les caractères saisis au clavier et affichés à l'écran (chiffres, lettres, ponctuation...). Si vous ne souhaitez pas modifier la police de caractères, vous pouvez uniquement modifier les caractères à partir de l'index 129.*

Assemblage de symboles

Pour rappel un caractère est contenu dans une matrice de 8x8 pixels. Mais comment faire si l'on veut un graphisme plus grand ? Pour cela, nous utiliserons un assemblage de symboles.

Rond comme un ballon, aussi jaune qu'un citron...

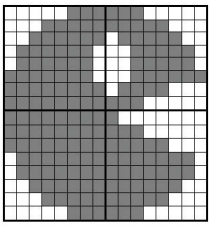
Nous allons réaliser un programme BASIC permettant d'afficher un PAC-MAN de 16x16 pixels en nous servant de l'assemblage de symboles.

Réinitialisez le CPC avec le code suivant :

```
CALL 0
```

La commande CALL permet d'appeler un sous-programme en langage machine à partir de l'adresse mémoire donnée. Ici, l'adresse 0 permet de réinitialiser complètement l'ordinateur.

Pour dessiner notre PAC-MAN, nous avons besoin de redéfinir 4 caractères, qui se présentent ainsi :

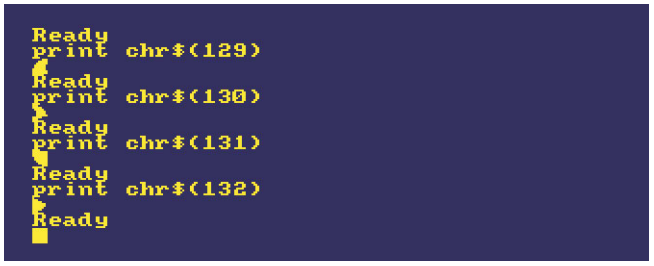


Afin de ne pas modifier les caractères saisis au clavier, nous allons redéfinir les caractères 129, 130, 131 et 132. Saisissez le programme suivant :

```
10 MODE 1
20 SYMBOL AFTER 128
30 SYMBOL 129,7,31,printadr,126,126,254,255,255
40 SYMBOL 130,224,248,124,62,62,63,248,224
50 SYMBOL 131,255,255,255,127,127,63,31,7
60 SYMBOL 132,128,224,248,254,254,252,248,224
RUN
```

Vérifions que nos caractères ont bien été modifiés en saisissant :

```
PRINT CHR$(129)
PRINT CHR$(130)
PRINT CHR$(131)
PRINT CHR$(132)
```



La commande CHR$ permet d'utiliser le caractère N de la table de caractères du CPC. Ici, on affiche nos 4 caractères redéfinis.

Assemblage !

Utiliser le code ci-dessus à chaque fois que l'on veut afficher notre PAC-MAN n'est pas vraiment pratique. Nous allons donc assembler les morceaux de notre PAC-MAN. Saisissez à la suite du programme, les lignes suivantes :

```
70 PAC1$ = CHR$(129)+CHR$(130)
80 PAC2$ = CHR$(131)+CHR$(132)
RUN
```

Les lignes 70 et 80 initialisent 2 variables PAC1$ et PAC2$. Le typage des variables est très simple en BASIC Amstrad. Le « $ » indique que la variable est de type « chaîne de caractères». Pour les nombres, il n'y a pas de symbole. Une variable ne peut pas avoir plus de 8 caractères.

PAC1$ assemble les caractères 129 et 130 l'un à coté de l'autre. PAC2$ fait de même pour les caractères 131 et 132. Saisissez les lignes suivantes :

```
PRINT PAC1$
PRINT PAC2$
```

Nous constatons que nos 2 variables affichent les parties haute et basse

de notre PAC-MAN. C'est mieux, mais on peut faire encore plus avec l'utilisation des caractères de contrôle. Nous allons créer une 3ème variable qui assemblera le tout. Saisissez ces lignes à la suite du programme :

```
90 PACMAN$ = PAC1$+CHR$(8)+CHR$(8)+CHR$(10)+PAC2$
RUN
```

La ligne 90 initialise la variable PACMAN$ avec nos 2 précédentes variables, assemblées grâce à 2 caractères de contrôle 8 (déplacement du curseur texte vers l'arrière) et un caractère de contrôle 10 (déplacement du curseur texte vers le bas). Voyons comment s'affiche désormais notre héros :

```
PRINT PACMAN$
```

L'affichage de notre héros est simplifié par la variable PACMAN$ qui assemble les symboles de caractères 129, 130, 131 et 132 d'une manière précise. Cette technique d'assemblage est très utilisée dans les jeux en BASIC. Mais l'exécution reste cependant lente pour afficher un assemblage plus grand. De plus, nous travaillons avec des caractères et sommes donc contraints d'utiliser les coordonnées texte et non graphique.

PAC-MAN version Sprite

Un sprite est un élément graphique, stocké en mémoire qui est dessiné sur l'écran. Sur CPC, l'affichage d'un sprite doit faire appel à une routine en assembleur afin d'être rapidement exécuté. Il serait possible de le faire en BASIC, en dessinant point par point. Mais la lenteur de l'interpréteur rendrait le programme inexploitable. Saisissons le programme suivant :

```
NEW
10 MEMORY &3FFF
20 REM Données du sprite (16 lignes de 4 octets)
30 DATA 0, 112, 224, 0
40 DATA 16, 240, 240, 128
50 DATA 48, 240, 112, 192
60 DATA 112,224,48,224
70 DATA 112,224,48,224
80 DATA 240,224,48,240
90 DATA 240,240,240,128
100 DATA 240,240,224,0
110 DATA 240,240,128,0
120 DATA 240,240,224,0
130 DATA 240,240,240,128
140 DATA 112,240,240,224
150 DATA 112,240,240,224
160 DATA 48,240,240,192
170 DATA 16,240,240,128
180 DATA 0,112,224,0
```

La ligne 10 réserve la mémoire après l'adresse &3FFF. Les lignes 30 à 180 contiennent les valeurs de chacun des octets de notre sprite. Chaque ligne du sprite est codée sur 4 octets (1 octet = 4 pixels de large en MODE 1). Notre PAC-MAN à donc une taille de 64 octets. Nous allons stocker notre sprite à partir de l'adresse &4000. Ajoutons les lignes suivantes à notre programme :

```
190 RESTORE 30
200 FOR ADR=&4000 TO &4000+63
210 READ A :POKE ADR,A
220 NEXT ADR
RUN
```

Explication du programme :

Ligne 190 : On branche le lecteur de DATA sur la ligne 30 de notre programme

Ligne 200 : Début de notre boucle. La variable ADR contient l'adresse mémoire à alimenter.

Ligne 210 : READ lit une valeur depuis la ligne DATA, la stocke dans la variable A et avance vers la prochaine valeur. POKE attribut la valeur de A à l'adresse mémoire ADR.

Ligne 220 : fin de la boucle.

Il ne se passe rien après l'exécution. C'est normal. Nous avons besoin d'une routine en assembleur pour afficher notre sprite. Pour cela, ouvrez la fenêtre Assembleur de WinAPE (touche F3). Créez un nouveau fichier et saisissez le code assembleur suivant :

```
org &6000
ld hl, &4000
ld de, &c000
ld bc, 16
affiche push bc : push de
ld bc, 4
ldir
pop de
ex hl, de ; call &bc26 ; ex hl, de
pop bc
dec b ; jp nz, affiche
ret
```

Assemblez le code en appuyant sur les touches Ctrl+F9. Notre programme en assembleur peut désormais être utilisé par le BASIC.

Revenez sur l'écran du BASIC et saisissez à la suite du programme BASIC les lignes suivantes.

```
230 CALL &6000
240 GOTO 240
RUN
```

Et voilà. Notre sprite apparait. A l'affichage, il n'y a aucune différence entre la version précédente du programme et la version sprite. Cependant, comme un sprite est un élément graphique, il peut contenir plusieurs nuances de couleurs et surtout être déplacé de manière plus fluide. De même, sur des éléments plus grands, l'affichage d'un sprite est largement plus rapide que l'utilisation de plusieurs symboles. Pour interrompre le programme qui boucle en ligne 240, pressez 2x la touche Echap. Améliorons notre sprite en lui apportant plus de couleurs, et pour le coup nous allons travailler en MODE 0. Saisissons un nouveau programme :

```
10 MODE 0 : MEMORY &3FFF:RESTORE 30
20 FOR R=&4000 TO &4000+ &40:READ A$:POKE R,VAL("&h"+A$):NEXT
30 DATA 00,CF,CF,00,45,CA,C0,80
40 DATA 45,C0,C0,2A,CF,D0,60,95
50 DATA CA,D0,E0,95,CA,D0,E0,80
60 DATA CA,D0,E0,00,CA,C0,80,00
70 DATA CA,C0,00,00,CA,C0,80,00
80 DATA CA,C0,C0,00,CA,C0,C0,80
90 DATA CF,C0,C0,95,45,C0,C0,80
100 DATA 45,CA,95,00,00,CF,CF,00
```

En MODE 0, notre sprite mesure 8x16 pixels. Les pixels étant de largeur double. Nous avons donc besoin de 1 octet pour afficher 2 pixels de couleur. Vous remarquerez que les valeurs contenues dans les lignes de DATA sont exprimées en hexadécimal. On aurait pu les écrire en nombre. C'est juste un confort de lecture.
La ligne 20 charge en mémoire les données à partir de l'adresse &4000. Nous allons à nouveau utiliser le code assembleur précédent pour affichage notre sprite. Ajoutons les 2 lignes suivantes à notre programme :

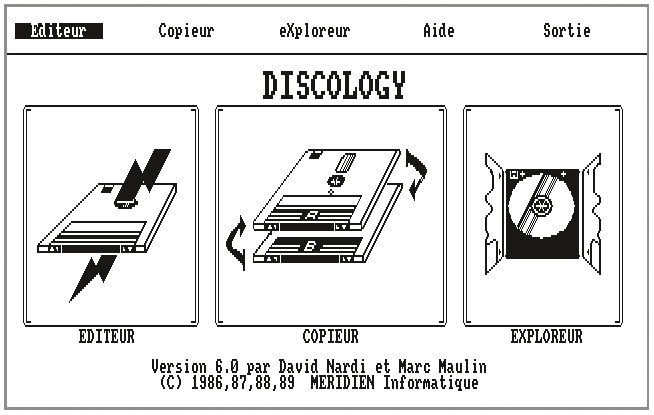
```
110 CALL &6000
120 GOTO 120
RUN
```

Notre PAC-MAN a pris des couleurs. Les pixels du MODE 0 sont plus larges.

Utilisation des différents modes d'affichage

Le MODE 0 permet de profiter de graphismes colorés mais manque de finesse. La plupart des jeux sur CPC utilisent ce mode.

Le MODE 1 offre des graphismes plus fins, mais moins colorés. Pour avoir des effets de dégradé ou donner le sentiment d'avoir plus de nuances de couleurs, le graphiste utilisait très souvent le principe de tramage, combinaison de pixels de couleurs différentes.



Le MODE 2 est surtout utilisé pour les applications (bureautique, utilitaires système...).

Il est possible, à l'aide de routines en assembleur, de partager horizontalement l'affichage en 2 modes différents. Comme ici, l'aire de jeu est en MODE 0 (16 couleurs), et le panneau de score en MODE 1. Il n'est cependant pas possible de superposer 2 modes différents ou de partager l'écran verticalement.
Certains programmes permettent l'affichage plein écran appelé « overscan ». La technique consiste à éliminer la bordure autour de l'écran et ainsi augmenter la résolution. Il s'agit de travailler sur les registres 1, 2, 6 et 7 du CRTC (Cathode Ray Tube Controller ou Contrôle du générateur de signal vidéo). La résolution finale atteint :
192 x 272 pixels en MODE 0.
384 x 272 pixels en MODE 1.
En BASIC il est possible d'obtenir le même effet en saisissant :

```
10 CLS
20 OUT &BC00,1 :OUT &BD00,50
30 OUT &BC00,2 :OUT &BD00,51
40 OUT &BC00,6 :OUT &BD00,34
50 OUT &BC00,7 :OUT &BD00,35
```

Visionnaire

En 1985, Alan Michael Sugar, PDG d'Amstrad explique sa vision de la micro-informatique :
« Les jeux c'est fini. Dans les années qui viennent, c'est sûr que ce marché va chuter et de beaucoup. Les gens cherchent quelque chose de plus sérieux, de professionnel... Je crois énormément au traitement de texte. » .
Heu.... (Source Amstrad Magazine N° 4 - Novembre 1985)

La gamme Plus

En 1990, Amstrad renouvelle la gamme des CPC en lançant la gamme Plus. Il s'agit de la même configuration (464 et 6128 uniquement), mais avec une carrosserie plus actuelle (plastique blanc) et une palette étendue à 4096 teintes (mais toujours 16 couleurs utilisables en MODE 0) et permet la gestion de 16 sprites hard (et zoomables). Cette gamme est aussi l'occasion pour Amstrad d'attaquer le marché du jeu vidéo (tenu par SEGA et Nintendo) en proposant la GX 4000, un CPC 464+ packagé dans une console de salon, auquel on a supprimé le lecteur de K7 et doté d'un support cartouche (support qui équipera également la gamme des CPC+). Les machines auront peu d'intérêt par rapport aux versions précédentes. La palette de 4096 couleurs n'est pas accessible par le BASIC, et le nouveau MODE 3 n'est qu'un MODE 0 en 4 couleurs qui ne sera jamais utilisé par les programmeurs. Malgré le soutien des éditeurs de jeux Européens (français surtout), la gamme disparaît des rayons à peine un an après sa mise sur le marché.

Où voir des CPC en action ?

La scène CPC est très présente en Europe, et plusieurs jeux indépendants sortent chaque année. La scène Espagnole est très active dans le développement de jeux. En France est organisée chaque année la démoparty ReSet (anciennement nommée Amstrad Expo) à Coutances, qui s'est déroulée en Juin cette année. On peut y rencontrer des passionnés du CPC, des concours de démos, des œuvres d'infographistes et des customs de machines. Pour en savoir plus, vous pouvez vous rendre sur cette page : http://reset.pushnpop.net .

Programmer en C pour CPC

Bien que l'assembleur soit le langage favori des CPCistes, il est possible de programmer en langage C et de compiler son code afin de l'exécuter sur CPC (émulateur ou vraie machine). Le compilateur C SDCC (Small Device C Compiler) fonctionne sous Windows et permet (entre autres) de réaliser des programmes pour Amstrad et de générer du code pour le Zilog Z80. (http://www.cpcwiki.eu/index.php/SDCC_and_CPC)

Découvrir ou redécouvrir le CPC

Logiciels d'époque
http://www.cpc-power.com
http://www.cpcgamesreviews.com

Tutorials de programmation assembleur et basic, articles techniques
http://www.cpcmania.com
http://www.cpcrulez.fr

AMSTRAD CPC 464, mon amour !

Après avoir fait mes armes en programmant des calculatrices (TI 57, FX 180P, PB 100), j'ai eu envie, tel l'extraterrestre du quartier, de faire l'acquisition d'un véritable ordinateur. A l'époque, tous ceux qui étaient proposés se ressemblaient beaucoup de par leurs possibilités, leur langage de programmation (le Basic) et leur prix. Compte tenu du fait que je devais le payer avec mes propres deniers, j'ai cherché un bon compromis entre prix et capacités. Mon choix s'est porté sur l'Amstrad CPC 464 couleur, qui à l'époque (1984) m'a coûté un mois de salaire de job d'été (650 euros en gros, oui mes jobs d'été étaient bien payés).

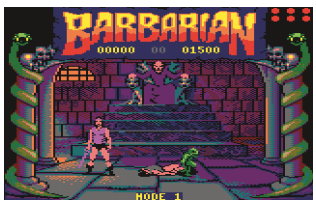
Stéphane Sibué
Directeur Technique chez GUYZMO
Microsoft MVP Windows Platform Development
www.guyzmo.fr
stephane.sibue@guyzmo.fr

C'était une vraie machine de course, processeurs Z80 à 4 Mhz, 64 Ko de Ram (kilo octets oui), clavier mécanique (très rare à l'époque) et lecteur de cassettes intégré, pas pour écouter de la musique, mais pour sauvegarder les programmes écrits en Locomotive Basic (un autre basic que celui de Microsoft). Ecran couleur intégré (une révolution) et un processeur sonore permettant de faire de véritables morceaux de musique par programme sans effort.

Jouer ou programmer ?

Généralement les gens faisaient ce genre d'acquisition pour deux raisons. Tout d'abord les jeux, qui étaient dignes des consoles de l'époque, et la programmation. Perso, vous le savez déjà, c'était pour la programmation, mais je dois avouer que j'ai aussi pas mal joué (j'avais 16 ans alors vous pensez bien). Le plus rigolo avec les jeux, du moins pour moi, ce n'était pas tellement d'y jouer mais plutôt de les pirater. Avec les cassettes, les premiers temps, il suffisait de posséder un bon copieur de cassettes et le tour était joué, les jeux étaient copiés en quelques minutes (oui minutes, l'unité de temps des chargements par cassette). Ensuite les éditeurs ont mis en place des stratagèmes pour éviter ce type de copie et il fallait en passer parfois par quelques lignes d'assembleur pour « casser » la protection, charger en mémoire les différents modules du jeu et les enregistrer ensuite sur une autre cassette, du grand art !

Le jeu Sorcery



Un must, le jeu Barbarian

Locomotive Basic

Coté programmation, l'Amstrad était servi par un basic de très grande qualité. Il était très rapide, et permettait d'utiliser toutes les ressources de la machine (sons, interruptions, ports, vidéo). Sur les 64 Ko de RAM, il ne restait que 42 Ko de disponible pour les programmes en basic, car il fallait enlever la mémoire utilisée par l'interpréteur et aussi la mémoire vidéo de 16 Ko. Coté vidéo c'était assez rustique. 3 modes étaient disponibles. Plus on voulait de couleurs différentes simultanées (dans une palette de 27 couleurs), moins on avait de pixels à l'écran :

Mode	Couleurs	Pixels	Caractères
0	16	160x200	20X25
1	4	320x200	40x25
2	2	640x200	80x25

Chose importante et très vintage, le Locomotive Basic était un Basic avec des numéros de lignes, comme tous les Basic de l'époque d'ailleurs. Quelle galère ces numéros de lignes quand j'y pense !
Mis à part ça, avec ce Basic, il était possible de faire des choses très sympathiques. Par exemple, il était possible de définir des interruptions au sein même du programme. S'il fallait exécuter un sous-programme qui se trouve en ligne 100, 2 fois par seconde, il suffisait d'écrire :

```
EVERY 5,0 GOSUB 100
```

Et toutes les 1/2 seconde (le 5 indique 5/10ème de seconde, soit 1/2 seconde), avec le timer 0, on saute automatiquement à la ligne 100. A la 1ère instruction RETURN rencontrée, le programme reprend là où il a été interrompu. L'Amstrad possédait 4 timers distincts (0 à 3) ce qui permettait de réaliser des choses bien pratiques très simplement. L'instruction AFTER, avec la même syntaxe, permettait de réaliser la même opération mais une seule fois et non cycliquement. Dans tous les cas, Il fallait juste bien penser à suspendre les interruptions (instruction DI) et les réactiver (instruction EI) aux bons endroits pour éviter des effets de bord induits par le fait que le programme pouvait être interrompu n'importe quand, c'est-à-dire dans n'importe quel état. Une très bonne mise en bouche pour la synchronisation d'applications.
Il était aussi possible de mélanger graphiques et caractères ou d'afficher des caractères sur des coordonnées graphiques. On pouvait scruter l'état du clavier, du joystick (indispensable pour jouer), imprimer, créer jusqu'à 8 fenêtres de texte différentes ayant chacune leur propre système de positionnement, lire et écrire des fichiers sur cassette (puis disquettes), charger des blocs de programmes au fur et à mesure des besoins, etc.

Le lecteur de disquettes, indispensable !

Très rapidement un investissement s'est imposé. L'achat d'un lecteur de disquettes, pour accélérer drastiquement les temps de chargement et profiter, oh comme c'est bon, des joies de l'accès aléatoire aux données (contrairement à l'accès séquentiel depuis une bande magnétique). Le lecteur de disquette était fourni avec un système d'exploitation très utilisé à l'époque, le CP/M. En gros, CP/M est l'ancêtre de MS-DOS. Il était utilisé principalement sur des machines professionnelles. L'intérêt a été pour moi de me familiariser en avance aux commandes typiques des systèmes d'exploitation de l'époque (DIR, COPY, REN). Amstrad fournissait aussi le DR Logo (DR pour Digital Research), un langage très riche et bien sympathique permettant de dessiner des figures géométriques (avec la tortue) mais pas que. Il était pourvu d'instructions permettant par exemple d'ajouter des propriétés supplémentaires à des « objets » préexistants et même de créer ses propres objets ; là aussi une mise en bouche pour la programmation orientée objet... avant l'heure. Grâce au CP/M j'ai pu aussi m'initier au Turbo Pascal de Borland qui était disponible (craqué, mais il y a prescription Monsieur le Procureur) pour cet OS.

Le lecteur de disquettes était aussi utilisable depuis le Basic, et franchement ça a changé complètement l'utilisation de mon Amstrad ! C'était de la folie, des disquettes d'une capacité incroyable de 178 Ko par face (eh oui, c'était trop de la balle), et surtout la possibilité de lire et d'écrire des données sans intervention de la part de l'utilisateur (plus besoin d'indiquer de mettre une cassette, de la caler au bon endroit et d'appuyer sur REC). Mon cher CPC devenait presque une machine de pro ! Les disquettes avaient un format qui a été abandonné par la suite, c'était des disquettes 3 pouces, mais elles étaient assez robustes ; 30 ans après j'arrive toujours à les lire !

Du basic à l'assembleur Z80

Très vite une communauté s'est créée autour de cette machine, des journaux se sont spécialisés dans la publication de programmes et d'infos la concernant, des éditeurs ont sorti énormément d'ouvrages permettant d'accéder aux ressources cachées de la bête (utilisation de routines en code machine depuis le basic via les instructions PEEK, POKE et CALL).

Le mensuel Amstrad magazine

La bible du programmeur

Toute cette littérature (seul média disponible à l'époque) a permis à beaucoup d'entre nous de progresser et d'aller de plus en plus loin dans l'utilisation de leur Amstrad.

En fait, mes débuts en programmation ont été un peu atypiques. Contrairement à beaucoup de personnes qui ont commencé à développer sur des calculatrices ou sur des ordinateurs personnels (Amstrad, Comodore 64, Oric, TO7, MO5, Amiga, etc), moi j'ai commencé directement par l'assembleur dès 12 ans, et tout ça à cause d'un concours de circonstances (je ne vais pas entrer dans les détails ici).

C'est tout naturellement qu'après l'utilisation d'un bon Basic ou d'un bon Turbo Pascal (langages de haut niveau), j'ai eu envie de retourner aux sources et de me remettre à développer en assembleur (non, ce n'est pas sale) pour tirer le plus de puissance de la bête. Donc je me suis mis à coder pour le Z80, le processeur de l'Amstrad (le même que celui de la Game Boy). Ce processeur de Zilog était très répandu à l'époque, et pour tout vous dire, on le trouve encore aujourd'hui dans les calculatrices programmables de chez Texas Instrument. Là aussi ce fut une belle expérience qui m'a servi très souvent dans mon travail par la suite.

```
 Program example1.asm      Line 1    Col 1    Free 31909  Insert
        org &4000
        nolist

        jp main

printstr:
        pop bc
        pop hl
        push hl
        push bc
printstrloop:
        ld a, ( hl )
        or a
        ret z
        call &bb5a
        inc hl
        jp printstrloop
L1:
        ret
printstrln:
        pop bc
        pop hl
        push hl
        push bc
```

Un peu d'assembleur ?

30 ans après toujours là

Il y a quelques semaines j'ai ressorti mon Amstrad de sa boîte, il a démarré sans broncher, 30 ans après son achat et au moins 10 ans après sa dernière mise en route. Puis j'ai branché son lecteur de disquettes, et là, catastrophe, impossible de lire la moindre disquette. Après un petit tour sur Internet, j'ai compris qu'il fallait changer la courroie d'entrainement. Un fois chose faite, il est reparti comme en 14 et j'ai pu montrer à mon plus jeune fils (18 ans tout de même) les jeux que j'avais et les programmes que j'avais créés, 15 ans avant sa naissance. Il a adoré et a même joué pendant un moment à certains jeux auxquels je jouais moi même. Si on m'avait dit ça il y a 30 ans...

Si vous aviez un Amstrad et que vous ne pouvez plus l'utiliser, il existe sur le net de très bons émulateurs. Personnellement je vous recommande le site WinAPE (http://www.winape.net/) dans lequel vous trouverez un émulateur Amstrad de grande qualité, pourvu de fonctions très puissantes et pratiques (je vous laisse découvrir).