



# 64BOPS

**NUMÉRO 1 • JANVIER 2023**

# ÉDITO

64 NOPs, ou l'improbable rencontre, 20 ans plus tard, entre *Amstrad Live*, *Quasar CPC* et *Another World*... Les plus anciens d'entre-vous se souviennent peut-être de ce projet, évoqué mais jamais concrétisé : la fusion des derniers fanzines en activité afin de donner naissance à un gros magazine collaboratif 100% CPC. Après des années et des détours, on se lance aujourd'hui dans l'aventure !

Dans ce premier numéro de rôdage, vous trouverez principalement des articles orientés technique et développement, ainsi que deux interviews. La ligne éditoriale n'étant pas figée, elle évoluera de numéro en numéro afin, pourquoi pas, de migrer vers un contenu plus généraliste (tests de jeux, courrier des lecteurs, *Amcharge*, etc.). L'objectif principal reste cependant la diffusion des connaissances techniques et l'aide au développement, quel qu'il soit (demo, jeu, logiciel, etc.).

Alors, comme dirait l'autre, « à quand le prochain ?! ». Dans un an sans doute, c'est l'objectif qu'on se fixe pour que tout cela n'empiète pas trop sur les projets CPC des rédacteurs... En attendant, vous avez 52 pages substantielles à ingurgiter, digérer, et exploiter pour vos futures prods : pas certain qu'une année suffise !

**Hicks / Vanity**



**4**

**BALADE DANS LE  
MONDE DE LA DEMO**

**10**

**INITIATION À L'UNIDOS**

**19**

**À TABLE !**

**24**

**AVIS AUX INSTABLES :  
PROGRAMMEZ EN TEMPS FIXE**

**28**

**CON VIBRATO**

**31**

**THESE COLOURS DON'T RUN**

**35**

**CODING TIPS**

**38**

**ASTUCES DE PUCEAUX**

**40**

**INTERVIEW : CNGSOFT**

**44**

**INTERVIEW : MAGE**

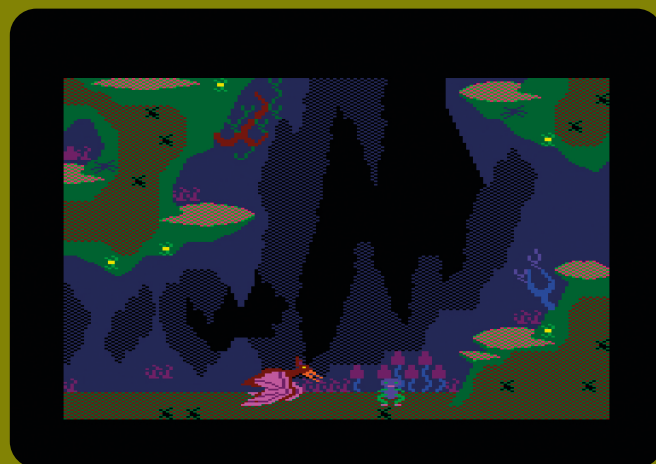
# BALADE DANS LE MONDE DE LA DEMO

1984, Roland is watching you. Euh, non Roland in the caves ! Vous incarnez dans ce jeu Amsoft une puce devant s'échapper d'une grotte en faisant des sauts de hauteur variable. Un sprite masqué sur un décor avant-gardiste se déplaçant grâce à un scrolling hard multidirectionnel... 1984 ! Roland lui-même ne s'en est pas remis. En programmant les registres 12 et 13 du CRTC, il est possible de déplacer des zones de la mémoire vidéo, sans effort.

PAR ELIOT / BENEDICTION

S'il y a un domaine dans lequel le CRTC a été grandement utilisé, c'est bien dans le demo-making ! Des scrolls hard dans tous les sens, des ruptures, des effets sur des ruptures "ligne à ligne" ( $R4=0$  et  $R9=0$ )... Mais, alors que la présence de sprites (soft) sur un déplacement (hard) est bien plus fréquente dans le domaine du jeu commercial que ce que beaucoup pensent, qu'en est-il dans celui de la demo ? Je vous propose dans cet article de faire une balade dans quelques demos affichant des sprites sur des écrans dont les offsets varient.

Pendant les premières années du demomaking CPC, l'utilisation du hard est assez rare, même si certains demomakers, souvent isolés, semblent vraiment en avance. J'ai cherché longuement et il semblerait que la première décoration soft sur un scrolling hard soit celle d'un self-made demomaker qui n'en est alors qu'à ses débuts ! En effet, dans *Overflow Tome 2*



Roland in the caves



Overflow Tome 2



sorti pendant l'été 1990, Croco 21 (alias Overflow) affiche plusieurs plans d'étoiles défilantes en mode 2 sur un scroll-text vertical descendant pour une fois et qui manifestement n'utilise pas le R5... C'est fin et on apprend dans le texte de la demo que ça a été programmé avec le *Hacker* !

Quelques mois plus tard, en octobre 1990, mettant à mal la réputation de son pays, le demomaker suisse TMP est bien moins précis dans sa partie nommée *Z80-Mania* dans la *Mc-Paddy 2* en affichant lui aussi quelques étoiles défilantes sur un seul plan au dessus du gros logo TMP se déplaçant horizontalement. Chaque étoile représente 2 pixels du mode 0 soit 1 octet qui n'est donc pas masqué avec le fond.

Mettant enfin le demomaking au niveau des jeux commerciaux tels *Mission Genocide* ou *Warhawk*, édités en 1987, Longshot réalise pour *The Demo*, sortie en 1991, un menu shoot'em up. Guidés par l'utilisateur, au premier plan, le vaisseau spatial et son tir permettent de sélectionner la partie à charger pendant qu'au second plan les 5 lettres du mot LOGON s'animent avec de jolies courbes ! Le troisième plan est constitué d'éléments graphiques de *Lightforce* et *Xenon*. Techniquement, une rupture classique assistée d'un décalage via le R5 permet le scrolling et l'alternance de 2 pages de 16Ko (« page-flipping ») permet d'afficher les sprites sans se prendre le balayage. Le reformatage de l'écran avec R1=32 simplifie le calcul de la position des sprites et leur affichage. Ne rigolez pas, la recette a rarement été améliorée depuis...

À la fin de cette même année 1991, Overflow marque un grand coup en sortant la *S&KOH* qui écrase tout au niveau des effets hardware et pour très longtemps ! Matérialisant l'arrivée des techniques de ruptures verticales, l'ondulation du grand dessin en Mode 0 de Zebigboss est même enrichie de différents sprites animés se déplaçant au premier plan. Il s'agit d'afficher le sprite sur un fond dont les



Z80-Mania



The Demo



S&KOH

adresses vidéo changent dynamiquement à chaque ligne. Spectaculaire pour l'époque mais finalement pas si compliqué à réaliser, ces sprites légendaires marquent surtout le sens du détail du demomaker et ça change tout...

Emboitant le pas de *The Demo* avec un menu de type jeu pour une megademo, en 1992, la *Paradise Demo* propose un sprite de chevalier

Paradise Demo



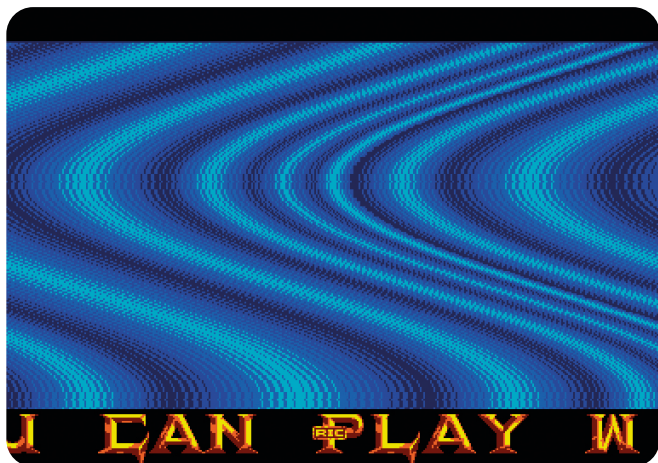
MegaParty'92



MegaParty'92



Sea You Soon



se déplaçant sur un scroll hardware horizontal. Les techniques de rupture et de scroll hard sont désormais répandues mais seul Tenebros alors âgé de 18 ans va plus loin dans sa partie en affichant un logo *Paradise Demo* composé d'une centaine de points ondulant sur un scroll hard vertical au format surprenant pour simplifier le calcul des adresses puisque  $R1=64$ ... avec  $R0=64$ . Ah, on me signale à l'instant une syncope chez Futurs'.

Dans la *MegaParty'92*, Odiesoft et Thriller nous offrent, eux, une très belle animation pré-calculée de 164 dots affichés devant un gros logo bondissant grâce au R5 et un changement d'offset. Un effet bien dynamique et qui ne profite pas, contrairement aux apparences, de la facilité d'un écran de 64 octets, l'écran étant formaté avec le standard  $R1=40$ .

Face Hugger, dans la même demo, nous présente un effet simple mais efficace, appelé *Snake* : l'affichage de sprites aidé d'un scrolling produit l'effet en lui-même. 4 boules sont affichées à chaque VBL puis l'écran de 16Ko monte de 8 lignes grâce à un changement d'offset sur un écran standard  $R1=40$ . L'effet de scrolling vertical est légèrement atténué par la présence de quelques étoiles qui ne subissent pas le scrolling.

Parmi les sprites qui (ne) se déplacent (pas) sur des scrolls texte horizontaux, il faut noter celui de Ric dans sa dernière demo *Sea You Soon* sortie en 1993. Un simple sprite figé de 4 octets sur 8 lignes, sans masquage. Bien statique tout ça... Oui, j'ai osé !

Dans *Die Kunst Der Demo*, 3 ans plus tard, Shap affiche un sprite non masqué également de 5 octets sur 10 lignes sur un gros scrolling texte se déplaçant avec une précision de 2 octets.

Avec un visuel assez proche de la partie de Tenebros, Squat affiche le premier objet en dots, non précalculé, se déplaçant sur 2 écrans ayant chacun un mouvement de



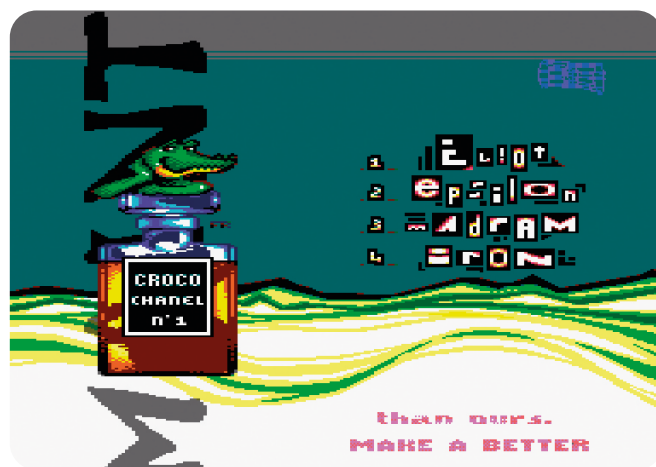
scrolling différent. Un bel exercice, malheureusement inachevé, pour cette preview diffusée en 1998.

Dans le menu de la compilation des meetings Croco Chanel sortie en 2002, Madram nous offre le café grâce à sa cafetière en 3D filaire en Mode 1 affichée sur un scrolling hard vertical déplaçant vers le bas un texte qui se poursuit en soft pour être déformé sur la bouteille de parfum en Mode 0. Un sprite qui décore un écran dans lequel se complètent scrollings soft et hard et se mélangent les modes de manière invisible.

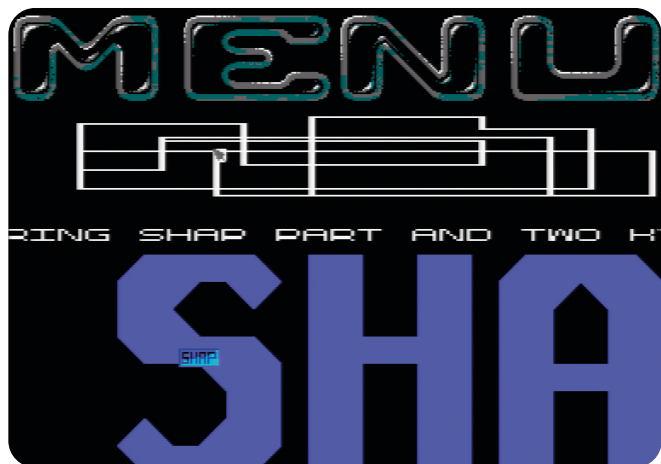
Un des aspects du demomaking est de tromper l'ennemi, comme le fait encore Madram dans la demo *Tire Au Flan* sortie en 2002. Alors que le cœur et le décor d'arrière plan paraissent fixes, il n'en est rien ! L'effet repose en effet sur un scrolling vertical de 8 lignes ! À chaque VBL, l'offset (R12/R13) est incrémenté de &30, le cœur est réaffiché à une nouvelle

position pour compenser le scrolling, le tout étant mystifié par une rotation de palette pour le fond et les dots, qui eux ne bougent pas... (ND : ma partie dans la *Croco Chanel* fonctionne sur un principe assez proche, avec un scrolling horizontal brutal et quelques éléments en premier plan...)

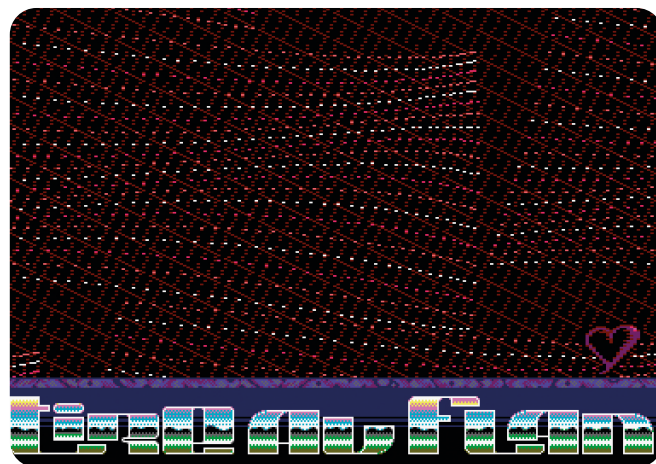
Dans sa partie *Critical Area* pour la *DemolzArt* dont la sortie a été retardée en 2005, Rainbird



Croco Chanel Demo



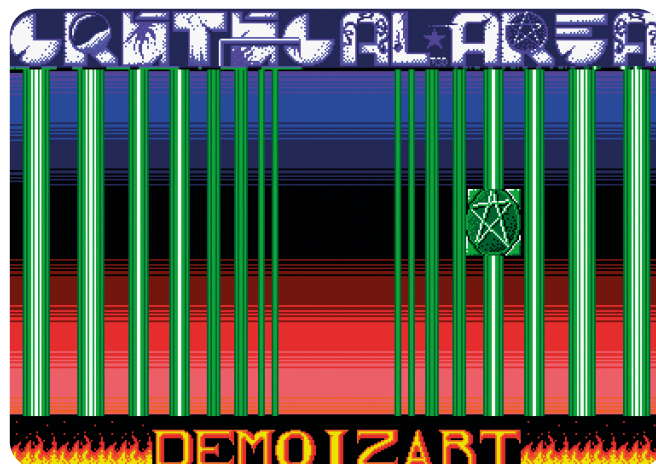
Die Kunst Der Demo



Tire au Flan



Squat's previews



Critical Area

déplace un sprite devant des barres verticales, produites par une rupture ligne à ligne classique ( $R4=0$  et  $R9=0$ ) permettant de répéter une seule et même ligne. Logée en &1614, la routine de Rainbird mène ainsi une course critique (!) contre le canon à électrons pour transférer à chaque ligne 8 octets en 47 NOPs et dessiner ce sprite. Astucieux mais il est possible de faire beaucoup mieux... Avis aux challengers !

Dans la megademo célébrant les 30 ans du CPC, sortie en 2016, Hicks propose un menu revisitant *Prohibition* permettant de déplacer un viseur sur un superbe décor de Beb et Barjack scrollant horizontalement et verticalement. Même si on en est encore loin, cela a pu raviver chez certains le fantasme du jeu sur scrolling multidirectionnel plein-écran ! À noter également dans ce menu, dans le scrolling texte que le haut des lettres en Mode 0 passant devant le nom des groupes est réalisé en soft, en complément du bas du scrolling qui lui

est en hard et en Mode 1, avec un motif d'arrière-plan rasterisé dont le pas de 4 pixels permet de paraître statique. Un autre exemple donc de mélanges de modes graphiques et de scrollings hard et soft pour tromper le spectateur.

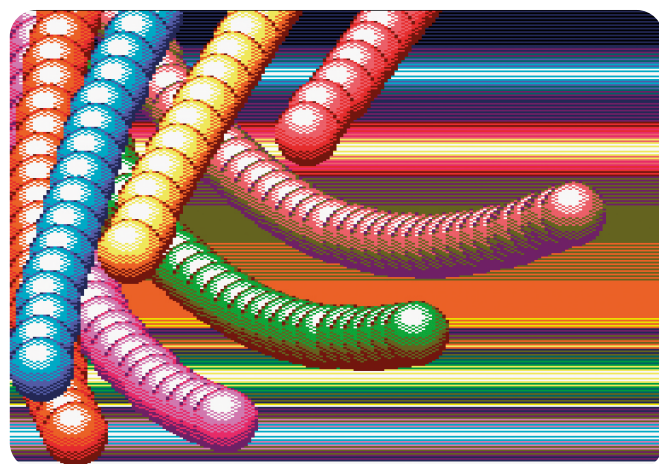
Dans les crédits de cette même megademo anniversaire, Krusty apporte un arrière-plan à son texte ondulant et scrollant verticalement, avec un champ étoilé vu de face. À chaque balayage, il s'agit de recalculer les positions des différentes étoiles car les offsets des 14 zones de texte varient.

L'esprit de tout demomaker étant de proposer toujours mieux, même 25 ans après, No Recess reprend et améliore l'effet de Face Hugger pour l'intégrer dans la très colorée *phX*, sortie en 2018. Scrolling plein-écran et plus de boules ! Cet écran a une structure bien particulière, tout en ruptures avec un  $R9$  mis à 5, permettant un scrolling plein-écran moins bru-

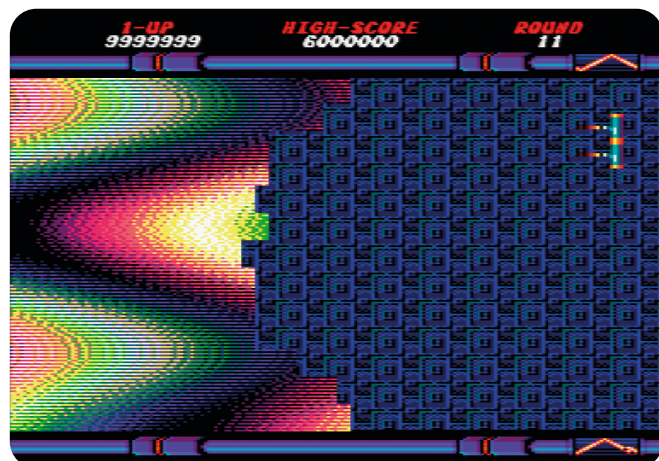
30 Years Amstrad Megademo



30 Years Amstrad Megademo



phX



Onescreen Colonies



tal que le classique saut de 8 lignes et donc une trace plus dense que celle de Face Hugger. Une prochaine étape dans l'histoire de cet effet pourrait être d'agir sur les boules affichées antérieurement ou d'ajouter un élément décoratif qui ne subit pas le phénomène de trace...

Dans la première partie de *Onescreen Colonies* imitant *Arkanoid*, Hicks déplace la raquette verticalement sur plusieurs ruptures verticales dont les structures évoluent au fil du temps. L'adresse d'affichage de chaque ligne du sprite dépend donc de la structure de rupture verticale à cette ligne.

Notre balade dans le monde de la demo s'achève maintenant, car le maquettiste me regarde avec ses gros yeux. D'autres produc-

tions auraient évidemment pu être évoquées ! Mon intention était de mettre en lumière les demomakers qui ont su apporter plus de relief ou de vie à leurs scrollings en rajoutant un plan ou une décoration ainsi que les astuces parfois invisibles ou méconnues des novices. Que retenir ? Tout d'abord, notons que les champs étoilés et les dots ont souvent été utilisés alors que l'on trouve finalement peu de sprites décoratifs devant des scrollings hard et ils sont relativement petits. Le mythe du demomaker en prend un sacré coup ! Ainsi, il reste encore à faire, car si afficher des sprites sur un fond mobile n'est plus un challenge technique en soi, cela peut amener de la profondeur, les écrans avec plusieurs plans étant assez rares sur CPC, surtout dès qu'il s'agit de ruptures complexes. Donc, sublimons nos prochaines demos, avec un peu d'imagination...

## Benediction Coding Party #2

Vingt-et-un ! Au moment du bouclage de ce magazine, je rentre tout juste du dernier et vingt-et-unième meeting organisé par Eliot, si j'ai bien compté. D'abord *Camembert Meeting*, se transformant en *Amstrad Expo*, puis en *Re-SeT*, et aujourd'hui en *Benediction Coding Party*, en passant par de ponctuels *Improvized Meeting*, *CPC at Work*, ou *CPC Meuuuhting*, sans compter les innombrables mini-meetings... Un changement de nom par décennie, en gros, et près de 25 ans de constance dans l'organisation de ces moments incontournables de la vie de la scène !



Alors pas de report pour *64 NOPs*, car il arriverait forcément trop tard, mais une photo souvenir et surtout un big up pour Eliot, dont le travail n'est pas assez remarqué, comme tous les CPCistes préférant œuvrer constructivement dans l'ombre plutôt que palabrer sur un énième groupe de discussion. Une abnégation et une constance qui devraient tous nous inspirer !

D'ailleurs, 2023 s'annonce riche en événements : *Glop Meeting* en février (sur invit'), *Revision* en avril, *BND Coding Party #3* en octobre, *Alchimie* en novembre...

Hicks

# INITIATION À L'UNIDOS

Depuis quelques années, il règne un petit air de chaos dans l'Univers du CPC. Sans crier gare, est sortie quantité de nouvelles extensions – souvent très prometteuses – qui permettent d'équiper nos vénérables machines de disques durs, de lecteurs de cartes MicroSD, de RAM disk, ou même de lecteurs USB ! Mais comme rien (ou presque) dans l'antique OS d'Amstrad n'était prévu pour accueillir des telles nouveautés, chacune de ces cartes s'est vue gratifiée de son propre logiciel de gestion ; ici une ROM DOS alternative, là une suite d'outils plus ou moins fantasques.

## PAR OFFSET / FUTURS'

Irrésistiblement, ces solutions logicielles sont venues avec leur propre ergonomie (souvent éloignée de celle de l'OS d'origine), avec leur propre jeu de fonctionnalités (manquant parfois de cohérence et d'homogénéité), et même si certaines ont véritablement prouvé leur efficacité, trop souvent, l'utilisateur exigeant s'est retrouvé confronté à des défauts de compatibilité plus qu'handicapants. Sans compter que, attisant la frustration, nombre de ces solutions logicielles se sont révélées être d'un usage exclusif ; qui a décidé d'utiliser telle interface se voit condamné à ne pouvoir utiliser telle autre en même temps.

Bercé du rêve de réconcilier tous les utilisateurs et tous les développeurs, de permettre à tous les logiciels prévus pour l'Amsdos de tourner n'importe où de la même manière, une petite ROM innocente a alors surgit du néant ! Son nom : *UniDOS*.



```
Amstrad 128K Microcomputer (f3)
©1985 Amstrad Consumer Electronics plc
and Locomotive Software Ltd.
UniDOS (integrated) ©2021 Futurs'
BASIC 1.1
Ready
█
```

L'*UniDOS*, c'est un système de ROM unifié et modulaire, un DOS qui est capable de prendre en charge toutes les interfaces existantes (et à venir !) tout en garantissant une compatibilité Amsdos aussi bien en terme de fonctionnalité que d'ergonomie. Grâce à un nouveau type de ROM appelées des « nœuds DOS », l'*UniDOS*



est capable de prendre en charge n'importe quelle interface, et n'importe qui peut coder un nouveau nœud DOS, rendant l'*UniDOS* extensible sans aucune contrainte.

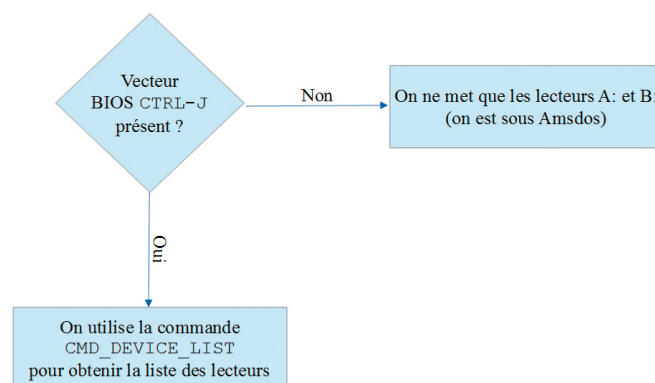
À ce jour, existent des nœuds DOS pour l'*Albireo* de Pulkotronics (support de la MicroSD card et du port USB), la *M4 Board* de Duke (support de la MicroSD, du DSK virtuel, des SNA, des CPR, et du FTP via le WIFI), la *X-Mass* de TotO et la *Symbiface II* de Dr. Zed (tous systèmes de FAT), ainsi que la *Nova* de Pulkotronics (mémoire non volatile). Tandis que l'*UniDOS* prend en charge directement les lecteurs de disquette (via la ROM Amsdos) et le lecteur de cassette (via les routines du Locomotive Software), les nœuds ajoutent donc chacun leur lot de lecteurs physiques au système, et bien sûr, tous peuvent être utilisés simultanément si plusieurs interfaces sont connectées à votre CPC.

Malgré cette abondance de lecteurs, du point de vue des vieux logiciels, rien ne change ; tous croient avoir affaire à l'Amsdos et à ses lecteurs A: et B:. Aussi, les noms de fichiers sont toujours au format 8+3, et il y a toujours des .BAS et des .BIN sauvés automatiquement pour les fichiers Basic et les fichiers binaires, etc. Avec l'*UniDOS*, le CPC reste un CPC. Mais derrière les apparences, l'*UniDOS* permet à tous ces programmes d'accéder à n'importe quelle interface (via les lecteurs physiques), de naviguer dans les répertoires (même s'ils ne savent même pas que la notion existe) et d'utiliser des liens symboliques (une fonctionnalité qui semble anodine mais qui permet des choses réellement étonnantes !). Il y a un petit côté magique à voir *OCP Art Studio* naviguer dans l'arborescence d'une clé USB ou n'importe quel crack de Chany ou des Crackers Velus se lancer depuis votre SD card comme s'il était encore sur la disquette pour laquelle il était prévu !

Je vois que le petit blond à lunettes commence à trépigner sur sa chaise. Il trouve cette introduction un peu trop longue... et il a raison ! Le

propos n'est pas ici de détailler le fonctionnement de l'*UniDOS* ; ce serait paraphraser la documentation disponible sur le site qui lui est consacré (<https://unidos.cpcscene.net>) où vous trouverez des explications détaillées sur tous les nouveaux concepts introduits par cette ROM : les lecteurs logiques et les lecteurs physiques (similaires à ceux que l'on trouvait dans la ROM *RODOS* de Romantic Robot), la gestion des chemins et les assignations, les liens symboliques, les RSX de manipulation des fichiers, etc., tout y est ! Non, ce que nous allons plutôt faire ici, c'est aborder l'utilisation de quelques-unes des API bas niveau, celles qui permettent à des logiciels nouveaux de tirer profit des avantages de l'*UniDOS* sans avoir à se soucier des extensions effectivement installées sur le système.

Tout ceci se fait via le vecteur BIOS CTRL-J qui permet également de détecter la présence de l'*UniDOS*. Ce vecteur offre une série de commandes (dont le nombre sera encore étendu à l'avenir) qui via lesquelles vous pouvez tout à la fois configurer l'*UniDOS* et accéder à ses fonctionnalités avancées. Pour ce premier article, nous allons nous familiariser avec les commandes permettant de récupérer les informations sur les lecteurs physiques ; fort utile à tout logiciel voulant offrir une ergonomie digne de ce nom à ses utilisateurs. L'algorithme pourrait être le suivant :



Ainsi, votre logiciel s'adaptera automatiquement au système de l'utilisateur, qu'il ait l'*Uni-*

DOS ou pas, qu'il ait une *Albireo* et une *M4 Board* ou une *X-Mass* et une *Nova*, il présentera toujours à l'utilisateur la liste des lecteurs effectivement disponibles sur son système.

Pour ce faire, c'est vraiment très simple, seulement quelques lignes d'assembleur sont nécessaires pour récupérer ladite liste de lecteurs physiques.

```
; Vérifie la présence d'UniDOS et
; configure le « far call » pour
; l'appel du vecteur BIOS CTRL-J de
; l'UniDOS.
```

```
; Ici, il s'agit simplement de
; chercher le vecteur BIOS CTRL-J
; qui est la signature de la ROM
; UniDOS.
; Une fois trouvé, on stocke
; l'adresse pour le « far call »
; qui permettra d'invoquer les
; fonctions avancées de l'UniDOS.
```

```
FindUniDOS
    ld hl,Command_J
    call KL_FIND_COMMAND
    ret nc
    ld a,c
    ld (FarCallAdr),hl
    ld (FarCallROM),a
    ret
```

Si cette routine retourne avec Carry=0, alors on devra se contenter de l'Amsdos et de ses lecteurs A: et B:. Sinon, si Carry=1, l'*UniDOS* est présent et on peut prestement récupérer la liste des lecteurs physiques à notre disposition via la commande `CMD_DEVICE_LIST`. Celle-ci attend en entrée dans DE un pointeur vers un tampon de 64 octets où sera stockée la liste, sous la forme d'une succession de chaînes de caractères dont chacune est terminée par le bit 7 mis à 1, tandis que la fin de la liste est marquée d'un 255.

```
; Récupère la liste des lecteurs
; physiques disponibles sur le
; système et la stocke dans
; BufferList.
```

```
; On appelle le vecteur CTRL-J avec
; la commande UniDOS
; CMD_DEVICE_LIST qui va remplir le
; tampon DE avec la liste des
; lecteurs physiques.
```

```
GetDeviceList
    ld c,CMD_DEVICE_LIST
    ld de,BufferList
    call CallCommandJ
    ret
```

La routine va là encore renvoyer Carry=0 en cas de succès et Carry=1 si la liste a pu être récupérée. Dans ce dernier cas, on peut alors par exemple simplement en afficher le contenu comme vous pourrez le voir dans le listing d'exemple complet ci-après.

Voilà donc comment en quelques lignes, vous pouvez rendre votre application dynamique afin qu'elle présente automatiquement à l'utilisateur la liste des lecteurs effectivement disponibles sur son système.

Pour aller un peu plus loin, vous trouverez aussi dans le listing d'exemple comment peut s'utiliser la commande `CMD_DEVICE_INFO` afin de récupérer quelques informations détaillées sur chaque lecteur. L'*UniDOS* vous permet en effet de savoir si le lecteur contient un média ou non, s'il gère les répertoires, s'il est protégé en écriture ou non, etc. Pour plus d'informations, je vous renvoie là encore vers le site de l'*UniDOS*.

```
; Exemple d'utilisation de quelques routines avancées de l'UniDOS.  
; Offset of Futurs'
```

```
; Quelques vecteurs système utiles.
```

```
TXT_OUTPUT      equ &BB5A  
SCR_SET_MODE     equ &BC0E  
KL_FIND_COMMAND equ &BCD4
```

```
; Quelques constantes pratiques.
```

```
CR equ 13 ; Carriage Return  
LF equ 10 ; Line Feed
```

```
; Commandes pour le vecteur BIOS CTRL-J de l'UniDOS.
```

```
CMD_CAT_FAKED_PARENT      equ 0  
CMD_DEVICE_LIST           equ 1  
CMD_DEVICE_INFO           equ 2  
CMD_DISABLED_LINK         equ 3  
CMD_EMULATE_SOFT_EOF      equ 4  
CMD_CAS_IN_READ           equ 5  
CMD_CAS_IN_SEEK           equ 6  
CMD_CAS_OUT_WRITE         equ 7  
CMD_CAS_OUT_SEEK          equ 8  
CMD_DISABLED_DIRECTORY    equ 9  
CMD_ENABLED_SORT_DIR_FIRST equ 10
```

```
; Notre programme commence ici !
```

```
org &8000
```

```
ld a,2                ; On passe en mode 2 pour faire de la place !  
call SCR_SET_MODE
```

```
call FindUniDOS        ; Est-ce que UniDOS est présent ?  
jr nc,UniDOSNotFound   ; Non, on rouspète !
```

```
call GetDeviceList      ; Oui, on récupère la liste des lecteurs.  
jr nc,GetDeviceListFailed ; Échec, on râle encore !
```

```
call PrintDeviceList    ; On affiche la liste des lecteurs récupérés.  
call PrintNewLine       ; On saute une ligne pour plus de visibilité.  
call PrintDeviceDetail  ; On affiche la liste détaillée !  
ret
```



UniDOSNotFound

```
ld hl,Msg_UniDOSNotFound
jp PrintString
```

GetDeviceListFailed

```
ld hl,Msg_GetDeviceListFailed
jp PrintString
```

GetDeviceDetailFailed

```
ld hl,Msg_GetDeviceDetailFailed
jp PrintString
```

; Vérifie la présence d'UniDOS et configure le « far call »  
; pour l'appel du vecteur BIOS CTRL-J de l'UniDOS.

FindUniDOS

```
ld hl,Command_J          ; Ici, il s'agit simplement de chercher
call KL_FIND_COMMAND     ; le vecteur BIOS CTRL-J qui est la
ret nc                   ; signature de la ROM UniDOS.
ld a,c                   ; Une fois trouvé, on stocke l'adresse
ld (FarCallAdr),hl       ; pour le « far call » qui permettra
ld (FarCallROM),a        ; d'invoquer les fonctions avancées
ret                      ; de l'UniDOS.
```

; Récupère la liste des lecteurs physiques disponibles sur le système  
; et la stocke dans BufferList.

GetDeviceList

```
ld c,CMD_DEVICE_LIST    ; On appelle le vecteur CTRL-J avec
ld de,BufferList        ; la commande UniDOS CMD_DEVICE_LIST
call CallCommandJ       ; qui va remplir le tampon DE avec
ret                     ; la liste des lecteurs physiques.
```

; Récupère les informations détaillées sur le lecteur physique  
; pointé par HL et les stocke dans BufferInfo.

GetDeviceInfo

```
ld c,CMD_DEVICE_INFO    ; On appelle le vecteur CTRL-J avec
ld de,BufferInfo        ; la commande UniDOS CMD_DEVICE_INFO
call CallCommandJ       ; qui va remplir le tampon DE avec
ret                     ; les informations sur le lecteur.
```

; Affiche la liste des lecteurs physiques stockés dans BufferList

PrintDeviceList

```
ld hl,Msg_GetDeviceListTitle
call PrintString
ld hl,BufferList
```

```

PrintDeviceListLoop          ; On parcourt la liste des lecteurs
    ld a,(hl)                ; en l'affichant.
    cp 255
    jp z,PrintNewLine        ; On finit avec un retour à la ligne.
    call PrintString
    ld a,' '                 ; Un petit espace entre chaque nom de
    call TXT_OUTPUT           ; lecteur.
    jr PrintDeviceListLoop

; Affiche la liste des lecteurs physiques stockés dans BufferList
; tout en récupérant les informations détaillées et en les affichant.

PrintDeviceDetail
    ld hl,Msg_GetDeviceDetail
    call PrintString          ; On parcourt la liste des lecteurs
    ld hl,BufferList         ; stockée dans BufferList, et pour
    ld de,BufferInfo         ; chaque lecteur on va demander les
PrintDeviceDetailLoop        ; informations détaillées dans
    ld a,(hl)                ; BufferInfo.
    cp 255
    jp z,PrintNewLine        ; On finit avec un retour à la ligne

    call GetDeviceInfo        ; Ici DE va être rempli avec les
    jp nc,GetDeviceDetailFailed ; informations sur le lecteur.

    call PrintString          ; On affiche le lecteur depuis BufferList.
    ld a,':'
    call TXT_OUTPUT
    ld a,' '
    call TXT_OUTPUT

    push de                  ; Puis on affiche quelques unes des
    push hl                 ; informations récupérées dans BufferInfo.

    ld hl,BufferInfo+10      ; Ici on a la description textuelle de
    call PrintString         ; notre lecteur physique.

    ld a,' '
    call TXT_OUTPUT

    ld a,(BufferInfo+0)      ; Là on a le registre de statut avec tout
    call PrintStatus         ; un tas d'informations utiles.

    pop hl
    pop de

```



```

call PrintNewLine          ; On revient à la ligne et on passe
                           ; au lecteur suivant.

jr PrintDeviceDetailLoop

```

#### PrintStatus

```

ld de,Msg_Media           ; Ici on décode les bits du registre
ld hl,Msg_NoMedia         ; de statut.
bit 0,a
call TestBit

```

```

ld de,Msg_Directory
ld hl,Msg_NoDirectory
bit 1,a
call TestBit

```

```

ld de,Msg_RO
ld hl,Msg_RW
bit 2,a
call TestBit

```

```

ld de,Msg_Removable
ld hl,Msg_Static
bit 3,a
call TestBit

```

```

ld de,Msg_StreamOnly
ld hl,Msg_Seekable
bit 4,a
call TestBit

```

```

ld de,Msg_UniDOS
ld hl,Msg_Legacy
bit 5,a
call TestBit

```

```
ret
```

#### TestBit

```

jr z,NoBit
ex de,hl

```

#### NoBit

```

push af
ld a,[' '
call TXT_OUTPUT
call PrintString
ld a,']'

```

```

call TXT_OUTPUT
pop af
ret

```

; Affiche une chaîne de caractères terminée par le bit 7 mis à 1.  
; Entrée: HL = adresse de la chaîne de caractères à afficher  
; Sortie: HL = adresse du caractère juste après la fin de la chaîne

```

PrintString
    ld a,(hl)
    res 7,a
    call TXT_OUTPUT
    bit 7,(hl)
    inc hl
    ret nz
    jr PrintString

```

```

PrintNewLine
    ld a,CR
    call TXT_OUTPUT
    ld a,LF
    jp TXT_OUTPUT

```

; Appel du vecteur BIOS CTRL-J via un « far call ».

```

CallCommandJ
    rst &18                ; Équivalent à un « rst 3,FarCallAdr »
    dw FarCallAdr          ; pour les assembleurs supportant la
    ret                    ; syntaxe des rst système Amstrad.

```

; Zone des données

Msg_UniDOSNotFound	db "UniDOS not found!",CR,LF+&80
Msg_GetDeviceListTitle	db "Device list:",CR,LF+&80
Msg_GetDeviceDetail	db "Device detail:",CR,LF+&80
Msg_GetDeviceListFailed	db "Failed to get device list!",CR,LF+&80
Msg_GetDeviceDetailFailed	db "Failed to get device info!",CR,LF+&80
Msg_Media	db "Media",&80
Msg_NoMedia	db "Empty",&80
Msg_Directory	db "Dir",&80
Msg_NoDirectory	db "File",&80



```

Msg_RW          db "RW",&80
Msg_RO          db "RO",&80

Msg_Removable   db "Removable",&80
Msg_Static      db "Static",&80

Msg_StreamOnly  db "Stream",&80
Msg_Seekable    db "Seekable",&80

Msg_UniDOS      db "UniDOS",&80
Msg_Legacy      db "Legacy",&80

Command_J      db &8a

FarCallAdr      dw 0
FarCallROM      db 0

BufferList      ds 64
BufferInfo      ds 64

```

Voici ce que vous obtiendrez en lançant ce petit programme d'exemple sur un CPC6128 équipé d'une *Albireo*, d'une *M4 Board* et d'une *X-Mass* :

```

Device list:
DFA DFB TAPE IDE M4 DSK FTP SD UMS ZERO

Device detail:
DFA: Floppy Disc A [Empty][File][RW][Removable][Seekable][Legacy]
DFB: Floppy Disc B [Empty][File][RW][Removable][Seekable][Legacy]
TAPE: Tape [Media][File][RW][Removable][Stream][Legacy]
IDE: IDE interface [Media][Dir][RW][Static][Seekable][UniDOS]
M4: MicroSD Card [Media][Dir][RW][Static][Seekable][UniDOS]
DSK: DSK Image File [Empty][File][RO][Removable][Seekable][UniDOS]
FTP: File Transfer Protocol [Empty][Dir][RW][Removable][Seekable][UniDOS]
SD: MicroSD Card [Media][Dir][RW][Static][Seekable][UniDOS]
UMS: USB Mass Storage [Empty][Dir][RW][Removable][Seekable][UniDOS]
ZERO: Bytes provider [Media][File][RO][Static][Seekable][UniDOS]

Ready
■

```

Il existe de nombreuses autres commandes via le vecteur BIOS CTRL-J qui ne sont pas utilisées dans cet exemple. Vous pouvez par

exemple modifier la façon dont est trié le catalogue en décidant si répertoires et fichiers doivent être mélangés ou non. Vous pouvez demander à y ajouter un indicateur de repertoire parent, désactiver complètement la gestion des répertoires et des liens symboliques, etc. Là encore, n'hésitez pas à vous référer à la documentation en ligne pour plus de détails.

Les commandes les plus intéressantes sont sans doute celles qui permettent un accès moderne aux fichiers (CMD\_CAS\_IN\_READ, CMD\_CAS\_IN\_SEEK, CMD\_CAS\_OUT\_WRITE et CMD\_CAS\_OUT\_SEEK), qui sont bien plus souples que celles fournies de base par l'OS d'Amstrad. Elles sont par exemple utilisées dans le player de fichiers VGM pour la carte *Willy+OPL3LPT* de *Pulkotronics* (<https://framagit.org/shinra/vgmplay>), mais comme semble l'avoir deviné le petit blond à lunettes, ce sera sans doute là le sujet d'un prochain article sur l'*UniDOS* !

# À TABLE !

Réalisation d'un sphère mapping étape par étape avec Rasm, simple, basique.

PAR ROUDOUDOU / RESISTANCE

## Vous n'avez pas les bases

Au moment où je commence le demomaking, il y a deux choses très en vogue : la déformation de texture sous quelque forme que ce soit et la 3D mappée. Bien que le rendu final n'ait rien à voir à l'écran, la plupart des effets qui en découlent utiliseront tous ou presque la même simplification vectorielle, à savoir passer par une grosse table de correspondance. Cette table peut prendre des tas de noms : displacement map, bump map, normal map, offset map, mais le principe reste le même. On a une grosse table qui va affecter des données et changer ou créer le rendu à l'écran. À noter que cette table fonctionne souvent de concert avec une texture (mais pas toujours !). Je vais aujourd'hui vous parler d'un effet qui à ma connaissance n'a jamais été réalisé sur CPC (hors prototype hyper simplifié plus proche du color cycling qu'autre chose...).



Prenons une routine de sprite, elle copie les données d'un tableau vers l'écran (qui est ni plus ni moins qu'un autre tableau de taille différente et ordonné différemment). Pour des raisons pratiques, le sprite est stocké de façon linéaire et on calcule à chaque ligne écran l'adresse de passage à la ligne suivante. C'est vraiment LA routine de base. Le code pourrait ressembler à ceci :

```

; HL=source
; DE=écran
; A=largeur
; XL=hauteur

sprite_vers_ecran
    push de
    ld b,0
    ld c,a
    ldir
    ex hl,de
    pop hl
    ld bc,#800
    add hl,bc
    jr nc,.pas_overflow
    ; BC=largeur écran-taille écran
    ld bc,80-#4000
    add hl,bc
.pas_overflow
    ex hl,de
    dec xl
    jr nz,sprite_vers_ecran
    ret

```

Lors de la copie avec le LDIR, nous avons pour chaque pixel une adresse source et une adresse destination, ces adresses sont calculées dynamiquement à partir des adresses respectives de départ / arrivée. Si on change l'adresse de départ, nous allons afficher un autre sprite (éventuellement le même, légèrement décalé). Si on change l'adresse d'arrivée, on change la position du sprite à l'écran. Simple, basique. Vous n'avez pas les bases.

Si on voulait se compliquer la vie, on pourrait stocker dans un tableau la liste de tous les offsets utilisés pour afficher notre sprite. Admettons que notre sprite commence en #4000, on aurait un tableau qui contient #4000, #4001, #4002, etc., vu que notre sprite est je le rappelle, stocké de façon linéaire. Le code deviendrait le suivant :

```

sprite_vers_ecran
    ld sp,hl
.copie_ligne
    pop hl      ; on récupère l'offset
                ; des données du sprite
    ld a,(hl) ; on lit la donnée
    ld (de),a ; on écrit sur l'écran
    inc de     ; pixel suivant sur l'écran
    ...
; gérer la largeur du sprite, le
; changement de ligne, le comptage de
; ligne...

```

C'est plus lent et pas très utile, mais pourquoi faire ça alors ? Je regarde ma boule de cristal.

Contrairement à l'idée répandue par un paquet de demomakers, une boule de cristal ne donne absolument PAS un effet loupe. Regarder à travers une boule de cristal n'est pas pratique, c'est un peu comme regarder dans un fish-eye qui inverserait l'image haut / bas / gauche / droite. Mais bon, comme on est demomaker, on va envoyer bouler les lois de la physique et faire un effet déformant. Au doigt mouillé, on aimerait que le milieu de la boule grossisse deux fois et que les bords de la boule réduisent deux fois. On irait de l'un à l'autre en respectant un sinus inverse à partir du centre de cette boule histoire d'avoir un effet bombé et non linéaire.

On me souffle dans l'oreille qu'il existe déjà sur CPC un effet du genre dans *L'Arche du Capitaine Blood* quand les planètes sont affichées. C'est vrai que c'est joli et au pixel, mais l'effet bombé n'est que sur l'axe horizontal et tout comme la preview d'Oliverflow, l'ensemble est statique. Il y a aussi un mapping statique dans le menu de la *Croco Chanel* sur la bouteille de parfum. C'est joli, mais nous irons plus loin dans cet article.



```

boule_carree
repeat 32,y
  repeat 32,x
    cx=(x-16-1)/16
    cy=(y-16-1)/16
    ; on norme notre vecteur à 4
    no=sqrt(cx*cx+cy*cy)*4
    ; on maximise la valeur
    if no>4 : no=4 : endif
    ; on utilise comme multiplicateur
    ; le carré de la norme, pour obtenir
    ; un effet "bulle"
    lowbyte=128+cx*no*no
    highbyte=hi(texturemap)+64+cy*no*no
    defb lowbyte,highbyte
  rend
rend

```

Il manque ici l'exclusion des zones hors de la sphère mais on va commencer par voir ce que donne ce carré en visuel avec un code d'affichage dédié et une texture dédiée. Comme on se promène dans les pixels, on ne peut plus les regrouper par deux dans un octet comme en mode 0, il faut un seul pixel par octet et on reconstruit la combinaison des deux à la volée. Par ailleurs, la texture va faire obligatoirement 256 pixels de large pour permettre de se déplacer facilement dans le sens de la hauteur, ce qui a son importance pour l'effet final.

```

ld sp,boule_carree
ld de,#C000
ld x1,32
.affiche_boule
repeat 16
  ; on traite les pixels 2 à 2 pour
  ; remplir les octets
  pop hl : ld a,(hl) : add a : pop hl
  or (hl) : ld (de),a : inc de
rend
ld hl,2*#800-16 : add hl,de
jr nc,.pas_doverflow
ld de,80-#4000 : add hl,de
.pas_doverflow
ex hl,de
dec x1
jp nz,.affiche_boule

```

Cette routine affichera toujours les mêmes pixels en provenance des mêmes adresses, ce n'est pas ce qu'on veut pour obtenir un effet visuel ! Mais comme nous avons organisé notre texture avec une ligne tous les 256 octets, il suffirait pour se déplacer à droite ou à gauche, d'incrémenter / décrémenter la valeur de la table et pour se déplacer de haut en bas, incrémenter / décrémenter le poids fort de l'adresse de la table. La boucle principale devient donc :

```

decale
  ; se déplacer en X à l'infini
  ld bc,255 : inc c : ld (decale+1),bc
  ld sp,boule_carree
  ld de,#C000
  ld x1,32
.affiche_boule
repeat 16
  pop hl : add hl,bc : ld a,(hl)
  add a : pop hl : add hl,bc
  or (hl) : ld (de),a : inc de
rend
ld hl,2*#800-16 : add hl,de
jr nc,.pas_doverflow
ld de,80-#4000 : add hl,de
.pas_doverflow
ex hl,de
dec x1
jp nz,.affiche_boule

```

## Je m'épuise car Thalès est toujours à faire

À présent nous avons une routine qui affiche une « bulle » carrée en haut de l'écran, il serait sympa de bouger cette bulle par dessus le graph de la texture, nous allons donc afficher à l'écran la texture et histoire de se faciliter une fois de plus la vie pour la suite, nous allons redimensionner notre écran pour qu'il soit carré :

```

; mode 0 + ROM OFF
ld bc,#7F80+%1100 : out (c),c
; visible word width
ld bc,#BC01 : out (c),c
ld a,32 : inc b : out (c),a
; HBL pos
ld bc,#BC02 : out (c),c
ld a,42 : inc b : out (c),a
; visible block number
ld bc,#BC06 : out (c),c
ld a,32 : inc b : out (c),a
; VBL pos
ld bc,#BC07 : out (c),c
ld a,34 : inc b : out (c),a

affiche_reference
; notre référence fait 256 pixels de
; large, notre écran seulement 128, donc
; on centre pour afficher de 64 à 192
ld hl,texturemap+64
ld de,#C000

ld x1,128
.affiche_reference
ld b,64
push de
.dispr
ld a,(hl) : add a : inc l : or (hl)
inc l : ld (de),a : inc e
djnz .dispr
; duplication des lignes
exx : pop de : ld a,e
ld hl,#800 : add hl,de : ex hl,de
ld bc,64 : ldir : exx

ld e,a
ex hl,de
ld bc,2*#800 : add hl,bc
jr nc,.pas_doverflow
ld bc,64-#4000 : add hl,bc
.pas_doverflow
ex hl,de
; increment dans le sprite
inc h : ld a,l : sub 128 : ld l,a
dec x1
jr nz,.affiche_reference
jr $
ret

```

## Les mathématiciens cachent leurs bornes sous un flot de formules

Maintenant, l'affaire se complique car si on souhaite déplacer notre boule sur le fond, il va falloir l'entourer d'une zone qui se contente de recopier le fond. Cette zone doit faire minimum une ligne en haut et en bas mais sur les côtés, comme nous avons deux pixels par octet, la zone doit faire un octet, donc deux pixels à droite et deux pixels à gauche. Cet embon-point non visible peut se gérer assez facilement en augmentant la largeur / hauteur de la table générée et en excluant du calcul les pixels trop loin du centre. On aurait quelque chose comme ça :

```

boule_carree
repeat BULLE_SIZE+2,y
repeat BULLE_SIZE+2,x
cx=(x-BULLE_HALF-2)/BULLE_HALF
cy=(y-BULLE_HALF-2)/BULLE_HALF
no=sqrt(cx*cx+cy*cy)

if no>1 || x<=2 || x>=BULLE_SIZE+1
; ça veut dire qu'on est hors du
; cercle, on met simplement la valeur
; "plate"
lowbyte=x-BULLE_HALF-2
; 64+y-BULLE_HALF-2
highbyte=hi(texturemap)+y-1
else
no*=sqrt(BULLE_HALF)
no*=no
if no>BULLE_HALF
no=BULLE_HALF
endif
lowbyte=cx*no
highbyte=hi(texturemap)+cy*no+BULLE_HALF
endif
defb lowbyte+BULLE_HALF+2,highbyte
rend
rend

```

Cette génération de table gère enfin les cas de recouvrement mais si nous voulons ultérieurement optimiser l'affichage pour suivre le contour de la boule, il faudra ne pas oublier le cas où la boule sera déplacée à la fois en vertical et horizontal. En effet, cela nécessiterait à chaque changement de largeur un octet supplémentaire à restituer. Avec notre map carrée nous n'avons pas à nous embêter à ce sujet, mais c'est plus lent ;)

Le code d'affichage complet (avec doublement des lignes en hauteur) devient :

## Les meilleures étudiantes préféreraient qu'on leur change les maths

Voici donc un premier effet visuel garanti qu'on peut obtenir avec une table d'offset mais comme j'en parlais en préambule, avec quelques changements mineurs pour peut réaliser une apparition progressive, un bump mapping, un tunnel ou même encore un roto-zoom ultra propre. Alors je vous dis peut-être à bientôt pour de nouvelles aventures.

```
.reference
    ld de,#C000 ; référence tout en haut à gauche de l'écran
; mise en cache dans les registres secondaires de l'adresse de la ligne suivante
    push de : exx : pop hl : ld e,l : ld a,h : add 8 : ld d,a : exx
.decalage
    ld bc,63
    ld sp,boule_carree
    ld x1,BULLE_SIZE+2 ; nous aurons une ligne en rab en haut
                        ; et en bas conformément à notre table
.affiche_boule
; ici je n'ai qu'un octet de rab car j'ai modifié légèrement la génération de la
; table pour tronquer la sphère sur les côtés (condition x<=2 et x>=bulle_size+1)
    repeat BULLE_HALF+1
        ; on traite les pixels 2 à 2 pour remplir les octets
        pop hl : add hl,bc : ld a,(hl) : add a
        pop hl : add hl,bc : or (hl) : ld (de),a : inc e
    rend
    exx
; doubler la ligne en hauteur, calcul de la ligne en dessous
    repeat BULLE_HALF+1 : ldi : rend
    ld bc,2*#800-BULLE_HALF-1 : add hl,bc : jr nc,.padov
    ld bc,64-#4000 : add hl,bc
.padov
    ld a,h : add 8 : ld d,a : ld e,l
    exx
    ld a,-BULLE_HALF-1 : add e : ld l,a : ld a,d : add #10 : ld h,a
    jr nc,.pas_doverflow
    ld de,64-#4000 : add hl,de
.pas_doverflow
    ex hl,de
    dec x1
    jp nz,.affiche_boule
```

# AVIS AUX INSTABLES : PROGRAMMEZ EN TEMPS FIXE

La scène CPC est pleine de gens instables, on le constate chaque jour. Gageons que cet article, dans lequel on recherche la meilleure façon de stabiliser le temps machine d'une routine, leur permette d'envisager l'avenir plus sereinement.

PAR HICKS / VANITY

On s'est tous amusés à matérialiser le temps machine que consommait l'une de nos routines par un petit raster dans le fond. Celui-ci est parfois fluctuant : en raison de code conditionnel ou du contexte, une routine peut prendre plus ou moins de NOPs. Il arrive que la situation soit gênante : lorsqu'on veut se positionner précisément sur l'écran, pour déployer des splits rasters ou une RVI, ou intercaler dans cette routine des OUTs quelconques à intervalles réguliers (toutes les 64 NOPs par exemple), la moindre variation de temps machine devient critique.

La question est alors : une routine étant donnée, quelle est la meilleure façon de stabiliser le temps machine de celle-ci ? Sachant que « meilleur » peut ici signifier :

- Comment le faire de façon optimale, en sacrifiant le moins de temps machine possible ?

- Comment le faire de façon efficace, en sacrifiant le moins de temps de développement possible ?

Les deux questions sont indépendantes mais également intéressantes : une excellente solution inapplicable est souvent moins utile qu'une solution grossièrement simpliste !

## Un exemple : les players de musique

Imaginons donc, pour la suite de l'article, que nous souhaitons stabiliser le temps machine d'un player de musique, plus exactement d'un streamer YM. Pour simplifier, le code consistera d'abord à décompresser X données, puis à envoyer ces mêmes données au AY. Dans la partie décompression, le temps peut fluctuer en fonction de ce qu'il y a à décompresser (disons, pour un LZ, une chaîne ou un caractère).



Dans la partie envoi AY, le temps peut fluctuer en fonction du nombre de valeurs ayant effectivement changé depuis la précédente frame.

On pourrait modéliser la situation avec un graphe orienté et pondéré, mais vraiment, j'ai essayé, et c'était trop moche. Je compte donc sur votre imagination, ou votre bloc-note. Le graphe est orienté car les branches de code ne peuvent s'exécuter que dans une seule direction, et la pondération représente le nombre de NOPs consommées par chaque branche. Les nœuds correspondent en gros à des instructions conditionnelles du type « JR Z,label ». On pourrait complexifier en incluant dans ces dernières des boucles avec un nombre d'itérations variable.

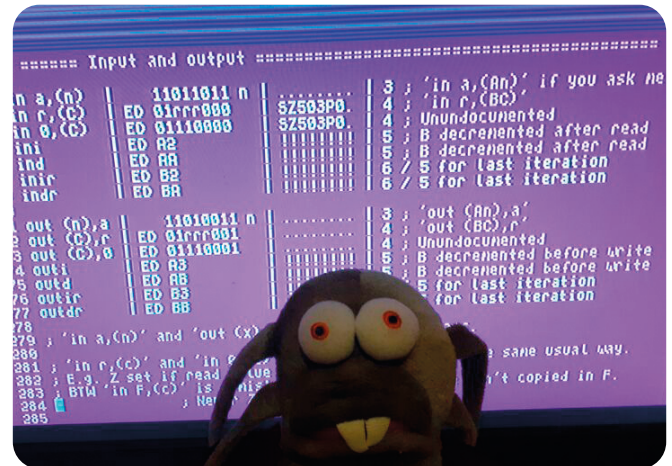
Un pré-requis pour la suite : disposer d'un tableau fiable des NOPs consommées par chaque instruction. Facile : vous trouverez sur l'exceptionnel site *64 NOPs* un article de Madram intitulé « Perfectly Accurate z80 flags and CPC Timings », issu de la notice d'*Organs*.

## La technique des maximums

Il s'agit de la plus évidente, de la plus facile à mettre en place, mais aussi de la plus mauvaise dès que le code devient long. L'idée est qu'en présence de plusieurs branches ou sous-branches parallèles, on aligne la durée de chacune d'elles sur celle qui dure le plus de temps.

On comprend rapidement que cela conduit à la pire situation possible : notre routine dure le temps du cas potentiellement le pire, même si celui-ci n'arrive jamais ! On se retrouve en quelque sorte en train de résoudre le problème du plus court chemin en théorie des graphes, mais inversé : identifier le pire chemin ! Ne comptez pas sur un thumb up de Dijkstra.

Dans le cas de notre player, cela reviendra à tout aligner sur le cas où on lit une suite de très petites chaînes avec un changement simultané de tous les registres AY. Un cas ex-



ceptionnel, qui n'a statistiquement qu'une faible probabilité de se produire. Que de temps machine gaspillé !

Mais soit : admettons cette première solution. Comment l'implémenter efficacement ?

Première approche : manuellement, soit la pire solution pour la pire technique ! Mais aussi la plus simple, donc la plus répandue. Il faut ici méticuleusement compter les NOPs de chacune des branches, identifier les branches parallèles, et réinjecter la différence pour égaliser les différentes durées. À chaque changement, il faut répercuter sur l'ensemble. Autant dire que ce n'est pas souple et que cela risque de conduire à des erreurs d'inattention dès que le code se complique.

Seconde approche : automatiser l'opération. Plus intéressant ! Plusieurs approches peuvent être envisagées, qui dépendent de vos outils. Le plus immédiat serait d'exploiter les directives d'un assembleur (*Organs* ou *Rasm*) : si celui-ci permet de retourner dans une variable le temps machine délimité par deux labels, il devient alors assez simple par quelques opérations arithmétiques de décider quelle est la branche maximale, et combien de NOPs ajouter dans les autres.

*Organs* inclut depuis la version FF (beta F) un tel compteur de NOPs, savamment médité par Madram et implémenté par toms (prof.o). Quelle fine équipe.

On peut donc envisager d'embarquer celui-ci dans notre source, et d'évaluer le temps machine de notre routine non pas à l'assemblage, mais avant son exécution, dans une phase d'analyse préalable. On exécute la routine dans les X configurations pertinentes, on range le tout dans un tableau, et on injecte le nombre de NOPs adéquat dans chaque branche.

## Un peu de dynamisme

La technique précédente a l'avantage de la simplicité, et peut suffire s'il s'agit de compenser une courte section de code. Mais dans des cas plus critiques, elle n'est pas digne de vous, homme de goût. Partons donc d'un autre principe : au lieu de stabiliser le temps machine *a priori*, faisons-le *a posteriori*. Concrètement, on laisse la routine s'exécuter normalement, sans stabilisation, en lui demandant de nous rendre, en paramètre, le temps qu'elle a consommé. Il semblerait que Madram ait pushé dans cette direction en même temps que moi, sans consultation préalable.

Un beau principe, mais comment l'implémenter sans que sa gestion ne soit plus lourde que la technique des maximums ?

Première approche : gérer un compteur de NOPs, auquel on ajoutera dans chaque branche le temps que celle-ci consomme. Solution simple et précise, mais qui a ses limites : comme on veut souvent compter plus de 256 NOPs, il faut allouer un compteur 16 bits, donc un registre dédié, ce qui alourdit les opérations. Si on parvient à libérer HL, alors un simple `LD BC,nb_nops : ADD HL,BC` fera l'affaire (6 NOPs). Mais dès qu'il s'agit de passer de nombreuses fois dans une boucle, on cumule lourdement ces additions. On peut donc envisager une seconde approche : travailler sur 8 bits, mais avec des unités de 2, 4, 8, 16 NOPS (ou plus) : dès qu'on a par exemple 8 NOPs dans une branche, on insère un `INC A`, quitte à arrondir en ajoutant des NOPs pour tomber juste. On perd quand même un peu de temps.

Dernière solution proposée : si le nombre de branches différentes est suffisamment réduit, allouer un compteur par branche (IXI, IXh, IYI, IYh semblent particulièrement indiqués) qu'on multipliera à la fin par le nombre de NOPs consommées par les branches auxquels ils sont associés.

À l'issue du comptage, il faut faire la somme, opération qui peut être coûteuse et qui doit être optimisée et déduite du temps machine disponible. Si, par exemple, la routine totale a consommé 2000 NOPs et qu'on veut stabiliser à 2048, il suffira d'attendre 48 NOPs, via une petite boucle.

## Macro-compensation

Le premier intérêt est d'éviter de se caler sur un maximum théorique peut-être jamais atteint en pratique. Et en même temps, comment prévoir la durée maximale que ne doit pas dépasser cette routine ? Bonne question. Vous pouvez soit l'anticiper, en estimant la durée maximale possible, ou l'évaluer empiriquement, avec des tests. Si c'est une musique, on peut imaginer qu'en amont, on la joue entièrement, qu'on place dans un tableau la durée que prend le player pour chaque itération, et ne garder finalement que la plus longue, sur laquelle on cale toutes les autres. Dans ce cas, ce n'est pas le maximum théorique des branches qui fait loi sur les autres, mais la somme des branches effectivement parcourues : on passe à un niveau macro.

Mais allons plus loin : cette routine ne sera sans doute pas la seule à consommer un temps machine variable. On peut donc imaginer faire la même chose pour d'autres routines (ne rien stabiliser mais compter les NOPs), totaliser ces NOPs, et n'effectuer la compensation qu'à la fin de chaque frame. On passe à un niveau encore plus macro, puisque les routines peuvent elles-mêmes se compenser, et s'équilibrer. À chaque fois, on augmente les chances d'auto-compensation. L'inconvénient, c'est qu'on peut moins facilement prévoir quels

seront les maximums si les routines ne bouclent pas en même temps.

Il devient alors possible, ô miracle, de se retrouver avec beaucoup de temps machine en rab à certaines itérations, et comble de l'optimisation : pourquoi ne pas alors anticiper sur la frame suivante ?

Pour faire ça bien, il faudrait un nouveau principe : pouvoir appeler une routine en lui donnant le temps machine qu'elle doit consommer. Typiquement, appeler une routine de décompression en lui demandant de décompresser pendant 512 NOPs. Ainsi, à chaque fois qu'un surplus de temps machine est disponible à une frame, il est immédiatement exploité et les routines peuvent se compenser d'une frame à l'autre ! Un conseil pour tous les adeptes de demos de boules, n'en déplaie aux puritains.

On est donc passé d'une méthode *a priori* (statique) à une méthode *a posteriori* (dynamique). Cette dernière se perfectionne en se généralisant : c'est d'abord au sein d'une routine donnée que le temps machine se compensait (level 1), puis entre plusieurs routines d'une même frame (level 2), puis entre plusieurs frames (level 3, finish him).

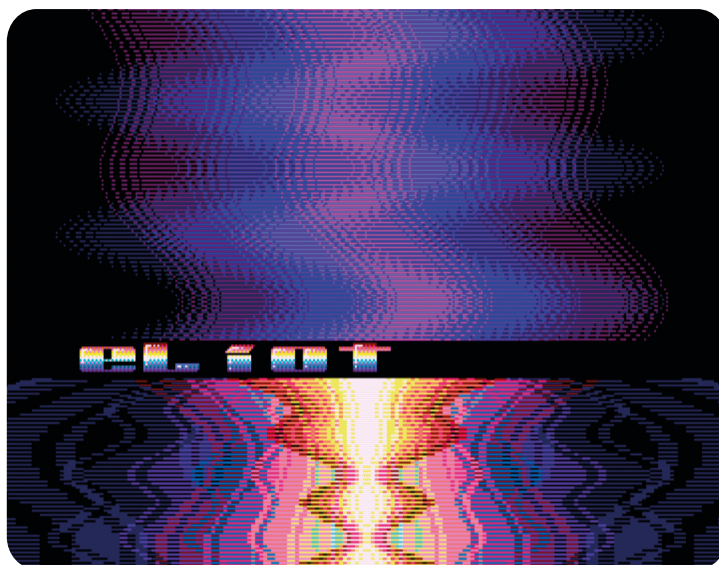
### Bonus : frame en temps fixe

Le beau Ramlaïd, qui nous manque, avait besoin pour la pénétrante *DTC*, d'un écran totalement stable en temps machine. Débuter des splits rasters avant le second HALT complique en effet la donne. Cela implique une stabilisation de toutes ses routines, mais aussi, chez lui, de supprimer l'attente VBL, et d'utiliser exactement 19.968 NOPs, pas un de plus ni de moins. Le petit brun du fond me fait remarquer avec un étonnant accent toulousain qu'il aurait pu tirer profit de la SHH (Synchro Half Halt) pour ajuster plus finement le début de la seconde interruption. Mais laissons cette scabreuse affaire de côté, qui fait encore couiner les puristes.

Avec un moyen d'automatiser la chose, c'est mieux, mais imaginez la galère si à chaque modification du code, il faut recalculer l'attente finale ! Trop d'abnégation.

Voici une alternative, utilisée et approuvée dans les subversives productions Vanity : pourquoi remplacer l'attente VBL (waitsync pour le style) par une suite de NOPs au lieu de l'utiliser pour ce qu'elle est, à savoir une boucle d'attente ? Votre code sera stable si lorsque vous entrez dans cette boucle, il reste précisément un temps multiple de la durée de cette boucle. Si par exemple il reste 600 NOPs inutilisées et que votre waitsync dure 6 NOPs, elle bouclera 100 fois. S'il reste 599 NOPs, à chaque frame, votre code sera décalé. La solution consiste donc à injecter en fin de source le temps restant modulo la durée du waitsync. Si on doit attendre 599 NOPs, on attendra d'abord 5 NOPs (599 modulo 6), puis on ira au waitsync qui bouclera 99 fois avant de rendre la main. Au pire, on doit donc ajuster manuellement une variable, qui prend une valeur entre 0 et 5, pour stabiliser la frame entière, ce qui est beaucoup plus pratique. Idéalement, il faut automatiser le calcul de cette variable via l'une des deux techniques présentées plus haut.

Vous pourrez ainsi rouler les mécaniques au prochain meeting, mais toujours dans le respect des conventions de Genève.





# CON VIBRATO

Ce titre n'est pas une insulte mais vous dévoile le sujet de cet article, sujet insolite suggéré par un grand ex-chevelu CPCiste qui s'intéresse à la musique sur nos vieux ordinateurs.

PAR ZIK / FUTURS'

Le vibrato est une variation rapide de la hauteur d'un son autour de sa fréquence de base. C'est un moyen d'ajouter de l'expressivité à une phrase musicale. On peut moduler son amplitude, sa vitesse de variation et la forme de cette variation.

Dans la musique instrumentale et chantée, le vibrato est diversement utilisé, selon l'époque : très légèrement dans la période baroque ou de façon appuyée en période romantique, et selon l'instrument : la clarinette classique de nos jours est jouée sans vibrato, contrairement à la plupart des autres instruments à vent.

Les musiques des jeux les plus anciens emploient souvent des sons simples, blancs et plats (par exemple *3D Stunt Rider* et *Commando*). Plus tard, des enveloppes de volume apparaissent et de très bonnes musiques, y compris récentes, n'emploient aucun vibrato

comme celles de *Deflektor* ou *Space Harrier*. Mais – lâchez ce coin de page – je vous prie malgré tout de poursuivre la lecture de cet article ! Pour vous en donner l'envie irrésistible, voici quelques exemples de musiques avec vibrato : *Cauldron II*, *Beyond the Ice Palace*, *Gryzor*, *Nemesis The Warlock*, *Warhawk*, *Saboteur II*, *Leviathan*, *The Abduction of Oscar Z*. Le petit blond à lunettes me cite *Back to the Golden Age*, mais heu... non.

Dans l'écriture d'une musique dans un tracker, on peut selon le cas penser le vibrato comme faisant partie du timbre de l'instrument, chaque note va exprimer ce même vibrato, ou comme une expressivité d'interprétation locale, en appliquant un effet à la note dans la pattern.

Mais alors, comment concrètement réaliser un vibrato avec le générateur sonore AY de nos CPCs ?



La fréquence d'un son définit sa hauteur, c'est-à-dire la note de la gamme et son octave. La période au sens physique est l'inverse de la fréquence ( $p=1/f$ ). Plus la période est grande, plus la note est grave. Les fréquences et périodes ont une nature logarithmique par rapport aux notes : la même note à l'octave supérieure voit sa période divisée par 2 (et sa fréquence multipliée par 2).

Pour faire jouer au AY la note voulue, on doit écrire dans ses registres la période (en réalité, la période au sens physique multipliée par un coefficient de 62500 et arrondie à l'entier le plus proche). Puis on va faire varier légèrement cette période pour ajouter du vibrato au son. Vous pouvez vous référer au chapitre 7.5 du manuel utilisateur (6.5 pour les Amstrad Plus) pour consulter la table des notes et périodes AY. Dans la suite du texte, j'utiliserai le numéro d'octave indiqué dans cette table, les logiciels utilisent des notations différentes.

A	110.000	568	-0.032%
A#	116.541	536	-0.055%
B	123.471	506	-0.038%

NOTE	FRÉQUENCE	PÉRIODE	ERREUR RELATIVE	
C	130.813	478	+0.046%	Octave -1
C#	138.591	451	+0.007%	
D	146.832	426	+0.081%	
D#	155.564	402	+0.058%	
E	164.814	379	-0.057%	
F	174.614	358	+0.019%	
F#	184.997	338	+0.046%	
G	195.998	319	+0.037%	
G#	207.652	301	+0.005%	
A	220.000	284	-0.032%	
A#	233.082	268	-0.055%	
B	246.942	253	-0.038%	

NOTE	FRÉQUENCE	PÉRIODE	ERREUR RELATIVE	
C	261.626	239	+0.046%	Octave 0
C#	277.183	225	-0.215%	
D	293.665	213	+0.081%	
D#	311.127	201	+0.058%	
E	329.628	190	+0.206%	
F	349.228	179	+0.019%	
F#	369.994	169	+0.046%	
G	391.995	159	-0.277%	
G#	415.305	150	-0.328%	
A	440.000	142	-0.032%	
A#	466.164	134	-0.055%	
B	493.883	127	+0.356%	

NOTE	FRÉQUENCE	PÉRIODE	ERREUR RELATIVE	
C	523.251	119	-0.374%	A international
C#	554.365	113	+0.229%	
D	587.330	106	-0.390%	
D#	622.254	100	-0.441%	
E	659.255	95	+0.206%	

Voyez par exemple le do octave -1 (que je noterai do-1), il a une période de 478. Le do0, une période de 239 et le do1, 119. Aux erreurs d'arrondi près, on divise bien par deux à chaque saut d'octave. En conséquence, si on regarde les différences en valeur, on voit que les notes de l'octave -1 sont comprises dans un intervalle de taille 239 et l'octave 0 un intervalle de 120, c'est-à-dire la moitié ( $478-239=239$  et  $239-119=120$ ).

À ce stade, on peut faire une première observation : pour réaliser un vibrato autour d'une note, il faudrait théoriquement s'écarter de  $1/n$  de ton au-dessus et  $1/n$  de ton en dessous de la note. Cette variation, une fois exprimée en période ou fréquence, représente un intervalle asymétrique, dont la largeur dépend de la hauteur de la note. Les trackers PC modernes comme *OpenMPT* ou *Renoise* gèrent cela très bien mais la précision du AY sur CPC nous amène à quelques compromis.

En effet, même des notes pas très aiguës ont une période AY aux alentours de 100, valeur relativement petite. Par exemple, le do à l'octave 1 (C1) a une période AY de 119 et les demi-tons supérieur et inférieur : 113 pour le do# et 127 pour le si. Si l'on voulait faire un vibrato aussi large (1 ton d'amplitude), on aurait comme variation -6 à +8.

Un vibrato en général est bien moins ample que ça. Prenons par exemple un tiers de demi-ton de part et d'autre de la note. Pour notre do1 centré à 119, ça donne une période de 117 à 122, donc une variation de -2 à +3. La dissymétrie que nous indique la théorie physique est bien là, mais elle est réduite.

Faisons maintenant un petit tour d'horizon des trackers CPC que je connais le mieux :

- *Arkos Tracker 2* ne propose pas d'option vibrato. Il faut soit utiliser une séquence de hauteur (pitch table) dans la pattern, soit écrire les variations de hauteur (period pitch) directement dans les lignes de l'instrument.

- *The Soundtrakker 128K* propose une option vibrato (option 7xy). x donne la vitesse de variation (par pas de 20 ms) et y l'amplitude ( $\pm 7$  maximum). La courbe générée démarre de l'aigu et fait un triangle. Par exemple on additionnera tour à tour à la période de base pour y=2 (et x=1) -2, -1, 0, 1, 2, -1, 0 en boucle (en réalité à un glitch prêt, il manque le premier pas au début de la séquence, donc la première hauteur quand x=1). Vous voyez que la vitesse du vibrato (la séquence complète) dépend de x mais aussi de y.

- Auto-promo : le *Soundtrakker DMA* a une option V similaire au *Soundtrakker* et offre aussi (gracieusement) des séquences de pitch.

Vous commencez à voir poindre la triste réalité : on fait très majoritairement l'impasse sur la dissymétrie et les vibratos employés à ma connaissance vont rarement au-delà de  $\pm 3$ . Et même assez souvent, on ne va décaler que dans un sens par rapport à la note, typiquement -1 en période. Vu la faible amplitude du vibrato, la forme de la variation est souvent un

sujet secondaire, atteindre la vitesse de vibrato voulue passe avant. On est limité par la période de replay, souvent de 20 ms (50 Hz).

Pour citer un exemple concret, les frères Follin utilisent fréquemment dans leur son lead aigu une séquence 0, -1, -2, -1, 0, 1, 2, 1 (*Gauntlet 3*, *Ghouls'n Ghosts*). Et aussi, le vibrato démarre avec un retard de 240 ms, c'est du plus bel effet ! Cette même séquence est utilisée sur Atari ST, ZX Spectrum et CPC alors que le générateur sonore n'est pas cadencé à la même fréquence sur ces trois machines (le CPC a la moins bonne précision des trois sur la période). Une conversion automatique n'aurait pas conduit à ce résultat. Si vous faites une écoute comparative, vous pourrez entendre cette différence sur les vibratos, plus exubérants sur CPC, d'autant plus que les versions ZX et CPC sont dans une tonalité 2 tons  $\frac{1}{2}$  plus aiguë que la version ST (si vous avez une explication à ça, ça m'intéresse).

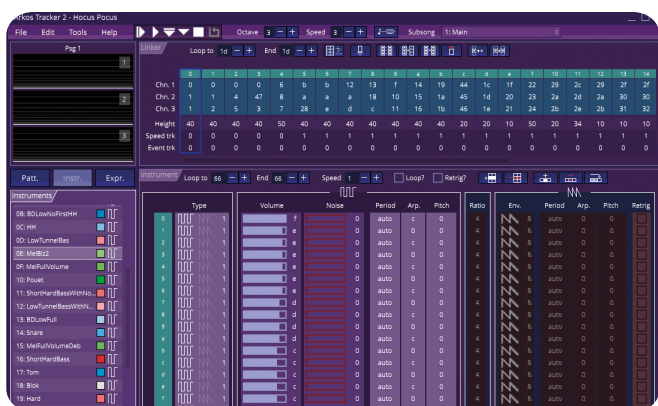
Est-ce que tous ces compromis sont un problème ? Et bien non, du moment que ça sonne bien !

La raison principale est que l'on compose avec les oreilles. On peut ajuster localement telle ou telle note qui sonne moins juste. Si le vibrato n'est pas symétrique, il faut veiller à ne pas trop décaler la note de sa fréquence de base, avec une attention particulière aux notes aiguës. En limite de résolution à l'aigu, on peut décider de simplement supprimer le vibrato pour les notes extrêmes ou user d'autres artifices comme des arpèges rapides (arpeggio) ou des notes glissées (portamento ou glide).

Dans le cas très particulier d'une musique générée, on voudra éventuellement faire les calculs de logarithme pour contrôler l'amplitude du vibrato en fonction de la hauteur de note. Plus raisonnablement, on ajustera l'amplitude par plage de périodes.

Nous voici arrivés à la fin de cet article qui j'espère vous aura intéressé et inspiré !

Arkos Tracker 2



Soundtrakker DMA



# THESE COLOURS DON'T RUN

En rédigeant les premières lignes de cet article, j'ai tout de suite pensé à cette phrase de Targhan dans *Demoniak #7* : « mes sprites, en fait c'est des split rasters. Ouais, j'ai réussi à optimiser le OUT à mort ». Il faisait probablement allusion au mystérieux OUT (n),A !

PAR TOMS / PULPO CORROSIVO

OUT (n),A consomme 3 NOPs (contre 4 pour un OUT (C),r), ce qui s'avère bien utile lorsqu'on a besoin de grapiller du temps machine. D'ailleurs une astuce bien connue des demo-makers est d'utiliser cette instruction pour mettre à jour plus rapidement le poids fort de l'offset (registre 12 du CRTC). Nous pouvons également tirer avantage de ce OUT pour

changer les couleurs de la palette. Par exemple, la demo *OneScreen Colonies #2* de Vanity l'utilise à merveille pour battre le record de split rasters sur une ligne. J'ai également exploité cette instruction pour gérer les magic rasters dans la end-part de *Debris*. Évidemment (sinon ce serait trop beau), son utilisation implique quelques contraintes que nous allons aborder.

## Comment ça fonctionne ?

OUT (n),A écrit le contenu du registre A à l'adresse An, ce qui veut dire que le périphérique adressé dépend directement de la valeur qu'on envoie sur le bus de données. Selon la combinaison, plusieurs périphériques peuvent être adressés simultanément (les adresses sont partiellement décodées sur CPC), ou pire encore, certaines valeurs ne peuvent pas être envoyées à certains périphériques. Pour y voir



plus clair, je vous invite à consulter le tableau suivant qui permet de comprendre comment sont décodées les adresses (version simplifiée du tableau publié par Grim sur le Grimware) :

PERIPHERIQUE	MASQUE D'ADRESSE
Gate Array	01xxxxxx xxxxxxxx
PAL	0xxxxxxx xxxxxxxx
CRTC Select	x0xxxx00 xxxxxxxx
CRTC Write	x0xxxx01 xxxxxxxx
CRTC Status	x0xxxx10 xxxxxxxx
CRTC Read	x0xxxx11 xxxxxxxx
Upper ROM	xx0xxxxx xxxxxxxx
Printer	xxx0xxxx xxxxxxxx
PPI Port A	xxxx0x00 xxxxxxxx
PPI Port B	xxxx0x01 xxxxxxxx
PPI Port C	xxxx0x10 xxxxxxxx
PPI Control	xxxx0x11 xxxxxxxx
Soft reset	11111000 11111111
FDC Status	xxxxx0x1 0xxxxxx0
FDC Data	xxxxx0x1 0xxxxxx1
FDC Motor	xxxxx0x0 0xxxxxxx

Première constatation : en mettant à 1 les 8 bits de n, on évite d'adresser le FDC ainsi que des périphériques tiers branchés sur le port d'extension. On utilisera donc dorénavant OUT (&FF),A.

Deuxième constatation, les numéros de couleur Gate Array étant compris entre &40 (%01000000) et &5F (%01011111), toutes les valeurs comprises dans cet intervalle adressent le Gate Array ! Mais pas que... le PAL et la ROM haute sont également adressés, ainsi que le port imprimante et le PPI par certaines d'entre elles. Arg !

Comme les choses sont bien faites, on peut facilement éviter d'adresser la ROM haute en utilisant le ghost register de INKR. Il suffit de mettre à 1 le bit 5 des numéros de couleur GA. On se retrouve donc avec des valeurs comprises entre &60 et &7F.

Ces valeurs n'étant pas des commandes valides pour le PAL (les bits 6 et 7 devraient être à 1 pour utiliser le registre MMR), celui-ci les ignorera tout simplement.

Voyons le reste en détail...

### Les périphériques adressés

Le tableau suivant indique les périphériques (autres que le Gate Array) adressés par chaque numéro de couleur GA :

INKR	PERIPHERIQUES
&60 0110 0000	Printer, PPI Port A
&61 0110 0001	Printer, PPI Port B
&62 0110 0010	Printer, PPI Port C
&63 0110 0011	Printer, PPI Control
&64 0110 0100	Printer, PPI Port A
&65 0110 0101	Printer, PPI Port B
&66 0110 0110	Printer, PPI Port C
&67 0110 0111	Printer, PPI Control
&68 0110 1000	Printer
&69 0110 1001	Printer
&6A 0110 1010	Printer
&6B 0110 1011	Printer
&6C 0110 1100	Printer
&6D 0110 1101	Printer
&6E 0110 1110	Printer
&6F 0110 1111	Printer
&70 0111 0000	PPI Port A
&71 0111 0001	PPI Port B
&72 0111 0010	PPI Port C
&73 0111 0011	PPI Control
&74 0111 0100	PPI Port A
&75 0111 0101	PPI Port B
&76 0111 0110	PPI Port C
&77 0111 0111	PPI Control
&78 0111 1000	
&79 0111 1001	
&7A 0111 1010	
&7B 0111 1011	
&7C 0111 1100	
&7D 0111 1101	
&7E 0111 1110	
&7F 0111 1111	



On constate d'emblée que les valeurs &78 à &7F adressent uniquement le Gate Array, ce qui veut dire qu'on peut utiliser ces couleurs sans aucun soucis. L'adressage du port imprimante n'étant pas un problème (débranchez votre *Soundplayer* !), nous pouvons également valider les valeurs &68 à &6F.

Penchons-nous maintenant sur les valeurs qui adressent le PPI, c'est maintenant que les choses se corsent (munissez-vous d'un bon schéma, celui publié sur *Quasar Net* par exemple).

Les valeurs adressant le port A du PPI (&60, &64, &70, &74) ne posent pas de problème à partir du moment où celui-ci est en sortie et que le PSG est en mode inactif.

Le port B étant par défaut en entrée, écrire sur le bus de données va provoquer un conflit PPI-Z80. Concrètement ça va juste mettre à jour le registre interne lié à ce port sans aucun autre effet. Il n'y a que lorsque le port est configuré en sortie que le registre interne est exposé sur le bus de données, pouvant générer des VBL parasites si le signal VBL est altéré, le PPI forçant celui du CRTC. Cependant l'électronique du CPC est tellement robuste que forcer les signaux fera tout juste chauffer le PPI (voir plus loin). On utilisera donc &61, &65, &71, &75 avec précaution surtout en période de canicule.

Le port C est encore plus problématique : &62, &66, &72, &76 configurent le PSG en mode lecture, nécessitant que le port A soit en entrée. Or, le PSG ne pouvant rester trop longtemps en mode non-inactif, il faudrait le

remettre en mode inactif immédiatement après le changement de couleur. Un non-sens puisqu'on utilise OUT (n),A pour gagner du temps machine !

Si le port A est en sortie, je cite OffseT : « le PPI et le PSG vont simultanément tenter d'imposer leurs niveaux de sortie l'un à l'autre. Ça crée un conflit électronique sur leurs lignes de données, un phénomène nommé "bus contention". Localement, ça va donc chauffer un peu (ça n'est sans doute même pas mesurable), le CPC va consommer un tout petit peu plus (tellement peu que ça n'est pas mesurable non plus), mais c'est tout ».

Ajoutez à ça que ces valeurs actionnent / arrêtent le moteur du lecteur cassette, sollicitant le relai interne des 664 et 6128. Celui-ci étant mécanique, il risque de lâcher s'il subit trop de transitions. On évitera donc d'utiliser les valeurs adressant le port C.

Enfin les valeurs adressant le port du registre de contrôle auront pour effet de mettre à 1 les bits 1 ou 3 de la valeur du port C, ainsi une ligne clavier quelconque sera sélectionnée. &63, &67, &73, &77 sont donc sans danger !

Les couleurs en « double » à la res-cousse

Je m'étais toujours demandé si on pouvait trouver une utilité aux couleurs en « double » du Gate Array (il en existe 5, avec des différences minimales mais difficilement observables). Eh bien oui ! Dans la plupart des cas, utiliser le double d'une couleur s'avérera beaucoup plus judicieux :

BASIC	INKR	PERIPHERIQUES	INKR	PERIPHERIQUES
1	&64	Printer, PPI Port A	&70	PPI Port A
7	&65	Printer, PPI Port B	&68	Printer
13	&60	Printer, PPI Port A	&61	Printer, PPI Port B
19	&62	Printer, PPI Port C	&71	PPI Port B
25	&63	Printer, PPI Control	&69	Printer

On remarque que les doubles adressent moins de périphériques simultanément. Toutefois, on préférera utiliser &60 à son double &61, c'est plus propre et Madram sera content.

Le tableau final

Voici la liste des valeurs qu'on peut utiliser sans risque avec OUT (&FF),A. En rouge, les couleurs déconseillées :

BASIC	INKR	PERIPHERIQUES
0	&74	PPI Port A
1	&70	PPI Port A
2	&75	PPI Port B
3	&7C	
4	&78	
5	&7D	
6	&6C	Printer
7	&68	Printer
8	&6D	Printer
9	&76	PPI Port C
10	&66	Printer, PPI Port C
11	&77	PPI Control
12	&7E	
13	&60	Printer, PPI Port A
14	&7F	
15	&6E	Printer
16	&67	Printer, PPI Control
17	&6F	Printer
18	&72	PPI Port C
19	&71	PPI Port B
20	&73	PPI Control
21	&7A	
22	&79	
23	&7B	
24	&6A	Printer
25	&69	Printer
26	&6B	Printer

Au final, on a accès à presque toute la palette, seules 5 couleurs sont à éviter !

Pour conclure, je tiens à remercier OffseT et Madram pour leurs éclaircissements sur certains points lors de la rédaction de cet article.

Un peu d'histoire (par Hicks)

« Mais alors si je comprends bien, Vanity et Pulpo Corrosivo ont inventé le OUT (n),A ?! », se demande le petit blond à lunettes, le boulet qui ne comprend jamais rien.

Une rumeur a plané pendant des années sur un éventuel usage du OUT (n),A dans l'intro de la *KKB First* (1990), fameuse pour son effet raster à première vue impossible sur le mot « SIE », qui semblait nécessiter un split raster de 3 NOPs. Mais Tom et Chilly sont plus rusés que cela : le « I » n'est en réalité qu'un morceau de border rasterisé, perdu au milieu de l'écran !



On trouve en revanche d'habiles OUT (&7F),A dans le code du menu de *Zap't'Balls* (1992), qui permettent à Elmsoft de passer trois encres au noir en alternant OUTs classiques pour la sélection de celles-ci, et OUTs « optimisés » pour positionner la couleur.



# CODING TIPS

Illustration de la solidarité entre codeurs, cette rubrique rassemble des astuces plus ou moins modestes glanées ici et là, au fil de discussions diverses. Partager, quelle idée !

PAR HICKS / VANITY

## RST Z,&38

Un soir d'été, le sieur Golem13, qui venait de ferrailler quelques heures de plus avec son ami Arthur, me livra un secret de conception. Lorsqu'on parcourt un écran dont l'offset évolue, on se contente souvent d'un piteux INC HL suivi de moults tests de bits pour gérer le bouclage. Or, celui-ci peut en réalité n'être vérifié qu'une fois sur 256, ce qui peut se traduire par :

```
inc l  
jr z,$+1
```

Si la condition n'est pas remplie (Z=0), on passe à l'instruction suivante (\$+2) sans sauter, et cela prend 2 NOPs. Si la condition est remplie (Z=1), ça saute en \$+1 en 3 NOPs, soit... à la donnée qui permet de coder le saut

relatif, ici &FF, qui se trouve être l'opcode de RST &38 ! Une donnée est donc utilisée comme opcode : Golem13 a inventé le RST Z,&38 ! 7 NOPs par appel (3 pour le JR, 4 pour le RST), retour classique par un RET.

## RST > CALL

L'astuce précédente permet de mettre en avant la très sous-exploitée instruction RST, qui supprime pourtant le CALL, sur plusieurs points :

- CALL : 5 NOPs (+3 pour le retour) et 3 octets,
- RST : 4 NOPs (+ 3 pour le retour) et 1 seul octet.

On ne dispose certes que de 8 RSTs, tous les 8 octets, de &00 à &38, mais on peut :

- Y loger des routines très courtes mais appelées très souvent (à l'exception du dernier, qui peut avoir la longueur voulue, à condition qu'on n'utilise pas les interruptions).
- L'utiliser pour interrompre sauvagement une routine soft pour y caser du hard, en ne pokant qu'un octet.
- À d'autres fins en se souvenant que RST écrit deux octets (en effectuant un PUSH du PC courant) et exécute un saut, avec certes des contraintes, mais pour seulement 4 NOPs ! Il y a sans doute là quelque magie noire à élaborer.

## LD HL,SP

Toms aime parfois faire basculer une adresse de SP à HL et vice-versa : peut-on le lui reprocher si les registres sont mutuellement consentants ? Le premier cas est résolu en 2 NOPs par l'étonnante instruction LD SP,HL, mais l'inverse n'existe pas. On se contentera donc de :

```
ld hl,0
add hl,sp
```

Pour remettre HL à 0 en 2 octets et 4 NOPs on peut tenter l'élégant SBC HL,HL (on s'assurera que Carry = 0). Certes un NOP de plus, mais un octet de moins : ça peut être bien utile pour réduire la taille d'un decruncher où le moindre octet compte.

## SBC, le ADD du pauvre

Des recherches archéologiques attestent de l'existence de la séquence suivante dans diverses productions :

```
ld bc,data16
or a
sbc hl,bc
```

Ce qui nous fait un total de  $3+1+4 = 8$  NOPs, oui ma petite dame. Certes, cela peut descendre à 7 NOPs en supprimant le OR A, si on sait que Carry = 0. Mais soyons raisonnables : soustraire un nombre revient à additionner son inverse, et les assembleurs gèrent depuis longtemps les nombres négatifs :

```
ld bc,-data16
add hl,bc
```

Soit  $3+3 = 6$  NOPs, sans condition sur Carry. Non mais.

## OUTD pas Dated

Qui a dit que OUTD ne servait à rien ? Sans doute pas toms, qui propose cette petite astuce pour initialiser la palette en 11 octets ("try to beat this!") :

```
ld hl,palette+&10 ; palette=&xx00
set_palette
ld b,&7f
out (c),l
outd
jr nc,set_palette
```

Qué Pasa ? OUTD commence par décrémenter B (qui passe à &7E, tout va bien), envoie la valeur contenue à l'adresse où pointe HL sur le port &7Exx, et décrémente HL. Tout cela est connu. Cependant, c'est un petit mic-mac au niveau de la mise à jour des flags. S'agissant du Carry, qui nous intéresse ici, il vaudra 1 si la somme de la valeur OUTée (qu'on appellera 'n') et de L (après décrémentation) dépassent 255.

Ici, puisqu'on met à jour la palette, les valeurs OUTées seront généralement comprises entre &40 et &5F. À chaque OUTD (sauf le dernier), on a donc  $n < 255$ , puisque même pour &5F avec  $L=\&0F$ , la condition  $\&5F + \&0F > 255$  est



fausse donc  $C = 0$ . Mais après le dernier OUTD,  $L = \&FF$ , donc la condition  $L + n > 255$  sera forcément vraie, ce qui permet de sortir de la boucle.

## Exploiter CPI

Dans la notice des instructions z80 d'*Orgams*, Madram livre bien des astuces, dont la suivante :

```
loop
    ...
    cpi
    jp pe,loop
```

CPI n'est pas une instruction très utilisée : elle équivaut à la série CP (HL) : INC HL : DEC BC, mais en 2 octets et 4 NOPs. Concrètement ici, ça va boucler tant que BC ne vaut pas zéro, ce qui est fort pratique en si peu d'octets. Pensez sizecoding.

## ADD signé

On se sent souvent obligé de travailler sur 16 bits dès qu'on manipule des nombres signés. Mais si l'une des grandeurs n'excède pas 255, on peut s'en passer et effectuer un magnifique  $HL = HL + C$  à la sauce Golem13 :

```
ld a,c
; glisse le bit 7 (= signe) dans Carry
add a,a
; si C, alors A=255 (nombre négatif),
; si NC, A=0 (nombre positif)
sbc a,a
; répercute le bit 7 de C sur tous les
; bits de B
ld b,a
add hl,bc
```

Si aucun signe n'est impliqué, on pourra se contenter de :

```
ld a,c
add l
jr nc,.no_carry
inc h
.no_carry
```

Soit 7 NOPs avec signe, et 5 sans : c'est honnête.

## Le délaissé INC (HL)

Les codeurs aiment les tables autobouclantes 8 bits. C'est bien légitime puisqu'on peut y naviguer en ne s'occupant que du poids faible. Typiquement :

```
pnt_table word table
...
ld hl,(pnt_table)
inc l
ld (pnt_table),hl
```

Soit 11 NOPs. Puisqu'on ne modifie que L, on pourrait simplement ne lire et n'écrire que le poids faible avec A, et tomber à 9 NOPs. Mais pourquoi passer par un registre et ne pas oser une incrémentation directe en 6 NOPs seulement :

```
ld hl,pnt_table
inc (hl)
```

## Astuce finale

Enfin, une astuce ultime : si aucun des conseils précédents ne vous est utile pour votre projet actuel et que celui n'avance pas depuis des mois, débutez un autre projet. Lorsque ce second projet patinera, débutez-en un troisième. Lorsque le troisième capotera, accusez votre graphiste. La technique est imparable.

# ASTUCES DE PUCEAUX

Мадрам, попробуйте в следующий раз написать чапо, это более практично для макета!

PAR MADRAM / OVERLANDERS ^ VANITY

## Façon foutrement efficace d'effectuer $A := n - A$

```
cp1      ; A := -1 - A
add n+1 ; compensate -1
```

## CP HL,BC (stable, mêmes flags qu'un CP classique)

Bien que n'étant pas le plus rapide, on appréciera la stabilité en temps-machine.

```
or a:sbc hl,bc:add hl,bc
```

Si le coeur vous en dit pas plus, vous pouvez emballer cela dans une MACRO. On peut omettre le OR A si on est sûr de l'absence de retenue.

(Parenthèse enchantée) Sans doute êtes-vous sûr au moment de l'écriture de l'état NC. Cependant cette supposition peut flancher au fil des remaniements de code, introduisant un bug. Aussi, je vous livre un conseil auxiliaire ; toute supposition de cette nature doit être vérifiée automatiquement. Ici, ça ressemblerait à :

```
fail = &be00 ; Breakpoint pour orgams
if dev_checks
    ; Expected NC for sbc below
    call c,fail
end
sbc hl,bc:add hl,bc
```

En cas de bug ou de doute existentiel, passer dev\_checks = 1 pour vérifier l'ensemble des suppositions.

## Pile, poil, valve, vulve

Généralement, l'utilisation de la pile pour lire / écrire dans des tables ne fait pas bon ménage avec les interruptions. Je prends ici ma casquette de conseiller conjugal, pour vous aider à mieux conjuguer.

### Lors de l'écriture :

- Remédiation 0 : écrire par adresses décroissantes. C'est sûrement déjà le cas, puisque dans ce scénario vous utilisez PUSH pour remplir votre table. Le RST &38 de l'interruption et éventuels PUSH (par la routine sous interruption elle-même cette fois) vont alors "corrompre" une partie de la table **non encore mise à jour**. On s'en moque donc, du rire gras et insouciant qui traverse la rue en dehors des clous.

Attention, la corruption devient réelle lorsqu'on atteint le début de la table. Prévoir une zone tampon placée devant la table (juste un 1 mot si aucun PUSH ni CALL).

### Lors de la lecture :

- Remédiation 0 : là encore, s'en moquer effrontément pour des entrées à **usage unique** (dans le sens où la table sera reconstruite entre temps) : les entrées corrompues sont celles qui viennent d'être utilisées, et vogue la galère.

- Remediation 0' : sans-doute êtes-vous une personne de goût (ou simplement nommée Goût), et votre entrée contient plusieurs mots dont seuls certains se voient rafraîchis. Auquel cas, alterner EI et DI, pour protéger les adresses sensibles. Quand le but est de ne pas (trop) décaler le déclenchement des interruptions et d'assurer la première interruption pendant la VSYNC, il suffit de ne pas rester en DI pendant plus de 32 lignes.

- Remédiation 1 : élaborant l'idée précédente, merde j'ai oublié ce que je voulais dire.

- Remédiation 2 : backup. Lors de l'interruption, on connaît l'adresse écrasée. On est donc en mesure de recalculer juste l'entrée en question. La routine sous int pourrait ressembler à :

```
int
; -- Sauvegarde contexte (pour éviter
; d'écraser d'autres entrées)
    ld (save_hl+1),hl
    pop hl
    ld (save_pc+1),hl
    ld (save_sp+1),sp
    ld sp,stack_int
    push af:push de

; -- Restore entry
    ld hl,(save_sp+1)
; Imaginons que la table soit
; en 4002-4fff, et qu'il y ait un
; backup en 5002-6fff
    dec hl
    ld e,l:ld d,h
    set 4,d
    ld a,(de):ld (hl),a
    dec de:dec hl
    ld a,(de):ld (hl),a

; -- Restore context
    pop de:pop af
save_hl ld hl,0
save_sp ld sp,0
    ei
save_pc jp 0
```

Évidemment, ce code peut se voir simplifié. Si le code interrompu n'utilise pas le registre A, nul besoin de sauvegarder AF, car F est inchangé. S'il n'utilise pas les registres secondaires, encore mieux, car on n'aura même plus besoin de sauver SP.

Mode ninja : certains registres n'ont pas besoin d'être sauvegardés, si les interruptions interviennent lors de sections soigneusement sélectionnées (EI temporaires). Là encore, très sujet aux bugs, on consolidera le code avec des assertions diverses.

# INTERVIEW : CNGSOFT

Personnage discret et polyvalent, CNGSoft a traversé toutes les périodes de la scène espagnole : des hits commerciaux pour hardcore gamers (*After The War*, *AMC*, etc.), au regain d'intérêt pour les homebrews depuis une dizaine d'années (4Mhz, ESP, Retrobytes Productions, etc.), en passant par l'arrivée fracassante de Batman Group en 2011. On revient aujourd'hui sur son impressionnant parcours.

PAR HICKS / VANITY

**Bonjour César. En préparant cette interview, j'ai réalisé que tu étais impliqué sur la scène CPC depuis la fin des années 80, ce qui fait de toi l'un des plus anciens CPCistes encore actif ! Peux-tu raconter tes débuts avec l'informatique ? Pourquoi avoir choisi le CPC ?**

Bonjour ; d'abord, c'est un honneur pour moi d'être invité à donner cette interview. Je ne me souviens plus des origines précises ; j'étais un petit gamin quand mon CPC 6128 arriva au cours de l'été 1988 (c'était une sorte de mode entre les parents qui avaient des idées vagues sur le futur proche de l'an 2000, devenu maintenant le passé) mais j'avais déjà lu une paire de livres sur le BASIC et les magazines de l'époque étaient riches en articles d'introduction à l'assembleur. Un peu comme tout le monde, mes premières aventures n'avaient pour objectif que de copier facilement sur

disque des jeux cassette ; petit à petit, le déplombage évoluait vers l'écriture de morceaux de code Z80 pour charger des fichiers, pour reloger les données, etc. 50% de cette évolution était instinctif, et l'autre 50%, expérimental.

**La scène espagnole est assez mal connue. Quels souvenirs de celle-ci gardes-tu ? Quels étaient tes contacts dans les années 80-90 ? Il semblait y avoir pas mal de crackers (The Spanish Hacker, Abraxas, Stell Mc Kraken, etc.), mais vraiment peu de demomakers (comme Kino & Hufo)...**

Je travaillais tout seul, mais il y avait une chaîne invisible d'échange de logiciels entre les utilisateurs, particulièrement les plus jeunes : le typique « un ami d'un ami d'un ami d'un ami a trouvé cette chose, es-tu intéressé ? ». C'est comme ça que je trouvais périodiquement des jeux déprotégés avec des



intros plus ou moins laides où les pseudonymes ridicules des anciens pirates (El Diodo ? Dalro-Mi ?? Despezuñadores anónimos sevillanos ???) étaient affichés avec des gros caractères en MODE 0. Naturellement, quelques intros étaient plus élégantes (par exemple le joli chien Collie de The Dog) mais le temps passait et en 1993 les 8 bits espagnols étaient commercialement morts. Mais j'insiste, c'était une chaîne invisible, je n'avais pas une vraie idée des sources, et je vivais trop loin du grand centre de la scène : le marché aux puces hebdomadaire de Madrid.

**Quels sont les jeux, demos, ou fanzines qui t'ont marqué à cette même période ? Ont-ils eu une influence sur tes activités ultérieures ?**

Il y avait un peu de tout, on ne pouvait pas se spécialiser : on avait peu de magazines de CPC pur, mais on pouvait apprendre du Z80 « sérieux » grâce aux magazines de ZX Spectrum, et même les livres pénibles de Data Becker (imprimés sans un seul atome de colle, les feuilles tombaient par terre) étaient un trésor avec leurs tables sur les opcodes du Z80 et leur description partielle du firmware. C'est pour ça que mon « modèle » était le Spectrum, pour les sciences et les arts : les titres classiques du Spectrum étaient le but de mes enquêtes, et je me souviens bien de la joie de trouver la version CPC de *Knight Lore*, une légende dans le monde du Spectrum, dont l'histoire était plus ancienne, profonde et prestigieuse. Seulement beaucoup plus tard j'ai découvert les merveilles du Commodore 64, et les liens historiques moins visibles (pas de Z80 pour partager le code avec le CPC) mais toujours authentiques : *Nebulus* et *The Sentinel* sont deux très bons exemples de jeux originellement conçus pour le C64 mais très bien adaptés pour les plateformes avec un Z80 au lieu du MOS 6510. Mais une fois de plus, la distance était un problème trop sérieux. Toute ma connaissance des fanzines et des demos était réduite à des bribes, des photocopies faibles de textes visiblement pas imprimés

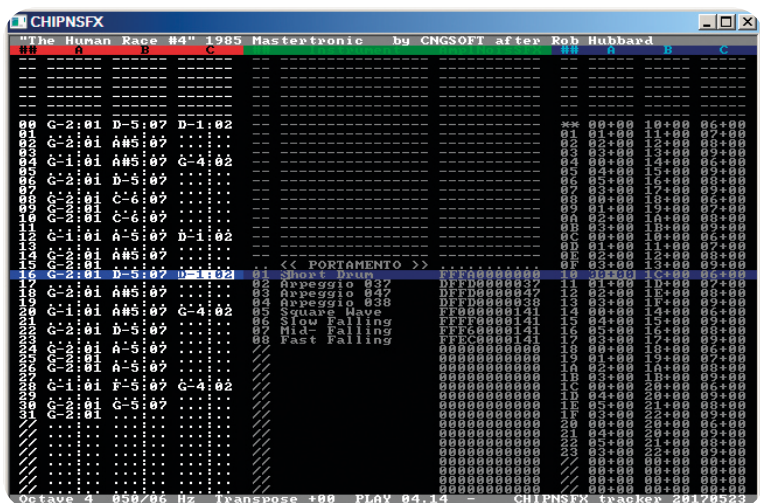
dans un vrai magazine, et des fichiers dans des disques échangés qui n'étaient pas des jeux ou des logiciels, mais « quelque chose » de différent, d'inclassable dans ces temps primitifs et mal communiqués, les « amis des amis des amis des amis » étant toujours la seule voie de communication.

**Une question que tout le monde se pose : pourquoi les jeux espagnols sont-ils si beaux mais si difficiles ?!**

C'est une question difficile, elle aussi. D'abord, on généralise trop fortement : la plupart des jeux espagnols n'étaient pas connus derrière les Pyrénées (la plupart des jeux étaient mauvais aussi, on ne doit pas oublier ça). La difficulté était trop souvent due à un manque de renseignements : le fameux *Abu Simbel Profanation* devient comiquement simple avec une bonne carte des écrans et les mouvements des ennemis. Il y avait aussi une différence démographique entre les marchés européens, avec plus d'enfants d'un côté, et plus d'adolescents de l'autre. Les plus jeunes avaient plus de patience, n'étaient pas facilement déçus par un seul problème ; ils jetaient plus de temps et d'effort sur les jeux. Les adolescents étaient moins tolérants et plus exigeants, plus sophistiqués. Par exemple, on ne voyait pas le moindre intérêt dans les jeux de rôle parmi les enfants et les 8 bits, mais les JDR avaient beaucoup plus de poids parmi les adolescents et les 16 bits. Mais je divague : le cœur du problème était la méconnaissance, et c'était un problème mutuel à travers les frontières. J'attends toujours de trouver quelqu'un capable de dire « *The Sentinel* est plus facile que *Navy Moves* » sans rougir de honte.

**Tu es l'auteur de *Chip'n'Sfx*, un logiciel de musique permettant de rejouer rapidement des musiques très compactes et minimalistes. Comment le projet est-il né, et sur quels principes repose-t-il ?**

Le point de départ est (une fois de plus) le déplombage. Quelques jeux (*Barbarian* de



Palace, Desperado de Topo Soft...) étaient plus gratifiants par leur musique que leurs graphismes et leurs mécaniques de jeu. Inévitablement, je me suis mis à les examiner pour trouver les routines de son, et c'est comme ça que j'ai compris qu'on peut écrire de la musique numérique avec des séquences d'octets pour les notes et les durées, les instruments de chaque note, les « call » aux morceaux de musique qui se répètent... Le cœur de *Chip'n'Sfx* (dont la première manifestation publiée - le « player » - est *Cngtro#1* parue au Ze Meeting 2003) n'est pas très différent. Chaque canal est indépendant et contient les données de la musique ou l'effet de son ; ces données sont un « byte stream » où les notes se mélangent avec les codes de durée des notes et les attributs du type de son (les définitions numériques des instruments), le tout avec quelques codes spéciaux pour faire des « call » et recycler les vers et les couplets utilisés plus d'une fois. Naturellement, *Chip'n'Sfx* a évolué depuis 2003, les codes spéciaux sont devenus plus riches et compliqués, le player vient maintenant avec un tracker - beaucoup plus ergonomique que d'écrire les données à la main ! - mais le cœur reste le même.

Chaque nouvelle version est accompagnée d'adaptation par tes soins de musiques de jeux au format *Chip'n'Sfx* (plus de 700 à ce jour !). Les adaptes-tu manuellement ou as-tu automatisé le procédé ? Le résultat est vraiment réussi !

Ah, merci beaucoup mais les choses deviennent beaucoup plus faciles lorsqu'on a un émulateur qui peut produire des logs du chip de son... Le but n'était pas la simple réplique des chansons ; je voulais trouver un point d'équilibre entre la conversion fidèle aux origines et l'adaptation simplifiée avec les instruments basiques de *Chip'n'Sfx* ; un équilibre qui pouvait être la meilleure preuve de la versatilité du logiciel. C'est alors une question de patience, et j'avais le temps (le seul privilège de la vie du chômeur presque chronique) et la volonté d'apprendre la logique de l'art de la musique. Après tout, s'il y a une façon d'apprendre rapidement la pratique des arts et des sciences, c'est l'analyse et l'imitation des travaux des maîtres. Malheureusement, ce n'est pas assez pour devenir un maître... mais au moins j'ai la possibilité d'improviser.

Chaque année, depuis son lancement en 2015, tu proposes un jeu au RetroDev organisé par Ronaldo pour l'Université d'Alicante. En général, le temps de développement est très réduit, comment procèdes-tu ? Quel conseil donnerais-tu aux développeurs de jeux, actuels et en devenir ?

Le procédé est chaotique. Ma vie quotidienne n'a pas d'espace pour la créativité, et quand le temps presse, le développement devient rapidement une chaîne folle de plagats et improvisations suivis de la désespérance la plus



profonde. C'est triste, mais quand on n'a pas un plan, tout se réduit à jeter des choses à l'écran jour et nuit (dormir n'est pas essentiel) jusqu'à la date limite : écrire la première version publique de *Epimetheus* en deux jours était simplement suicidaire. Naturellement, après cette histoire cautionnaire, mon conseil pour les développeurs est simple : ne faites pas comme moi !

**Justin (2005) et Hire Hare (2016) rendent un bel hommage aux jeux en 3D isométrique de l'époque. Envisages-tu de créer un jeu de plus grande ampleur basé sur ce principe ? Un *La Abadia Del Crimen 2* en quelque sorte !**

Oui, je rêve d'écrire un nouveau jeu en 3D isométrique ; mais un bon jeu, sans le primitivisme de *Justin* et la précipitation de *Hire Hare*. Je ne suis pas créatif, mais je veux dépenser plus de temps et de travail sur ce jeu hypothétique, et je crois que ceci est possible malgré toutes les complications qui m'assaillent tous les jours, toutes les heures.

**Tu es également l'auteur d'un émulateur CPC nommé CPCE. Quelle partie de l'émulation t'a posé le plus de problèmes ? Quelles sont les améliorations à venir ?**

L'ancien *CPCE* n'était plus qu'un étrange passe-temps écrit en assembleur i286 (plus tard, i386) et même s'il m'a bien servi pour beaucoup de choses, il était condamné à devenir insoutenable : l'assembleur moderne est trop compliqué à maintenir, les projets sont beaucoup plus grands qu'au bon temps du Z80, et la multiplicité des plateformes d'aujourd'hui est l'antithèse du travail de programmation spécialiste. C'est comme ça que nous avons maintenant *CPCEC* (dont la nouvelle lettre à la fin du nom vient du langage C) qui non seulement m'a donné l'opportunité de finir un épisode de ma carrière académique (le projet final du Master en Informatique en 2019) mais il a aussi ouvert des nouvelles portes et (plus important) possède un degré de fidélité

beaucoup plus fort (l'émulation au niveau NOP, et dans quelques parties, au niveau T) et des projets additionnels avec les mêmes matériaux : on a déjà *ZXSEC*, l'émulateur de ZX Spectrum ; *CSFEC* (Commodore 64) est en développement ; et le futur... ah, si on a le matos et le temps, on peut créer plus de choses au futur ! Mais à présent les choses sont plus limitées, avec le CRTC qui me donne toujours des migraines (*Overflow Preview 2* sur CRTC 0, *Living Daylights* sur CRTC 1...) et le FDC qui a besoin d'une émulation propre des délais : les « overruns » sont simulés « ad hoc », assez bien pour les protections, mais trop mal pour les demos qui lisent des données sans arrêter de jouer de la musique, par exemple *Batman Forever* et *PhX*.

**Cracker, demomaker, codeur de jeux, de logiciels, musicien, graphiste à tes heures... Tu auras exploré tous les aspects de la machine ! Une préférence, finalement ? Pourquoi ne pas avoir poursuivi tes activités dans le domaine du demomaking ?**

À la fin, je suis plus un codeur qu'un artiste ; je ne trouve pas l'équilibre entre les diverses activités et je m'écrase toujours contre le mur des graphismes et de la musique, et plus généralement de la création d'idées et de concepts qui sont la raison d'être de chaque logiciel, et encore plus quand on parle de demos. J'ai le « comment » mais pas le « quoi », et le premier ne peut pas exister sans le dernier.

**Merci d'avoir répondu à mes questions en délaissant ta langue maternelle. L'usage veut que je te laisse le mot de la fin...**

Je veux remercier les auteurs qui restent toujours actifs avec leurs demos, jeux et logiciels, qui maintiennent les 8 bits vivants. C'est grâce à eux que je reste toujours ici, éternellement fasciné par les merveilles qu'ils peuvent faire avec moins de 8 MHz et 256 kB, et je ne perds pas l'espoir de contribuer avec quelque chose d'utile ou de joli malgré toutes les choses qui me tombent dessus.



# INTERVIEW : MAGE

Figure emblématique de la scène demo des années 94-96, acolyte historique d'Antoine, Mage nous éclaire dans ce long entretien sur cette époque et sur son rapport actuel au CPC, qu'il redécouvre depuis quelques mois. Time 2 Go Back !

PAR HICKS / VANITY

**Bonjour Mage ! Peux-tu te présenter pour les lecteurs qui ne te connaîtraient pas ? Qu'est-ce qui t'a conduit vers le CPC ? Où as-tu appris le code et les rudiments du demomaking ?**

Salut Hicks :) Je m'appelle Arnaud, né en 1973 et j'ai eu un 6128 Azerty CRTX 0 quand j'étais en 4ème. J'avais touché auparavant un Atari 800XL et la programmation m'avait séduit. Donc arrivé sur CPC, j'ai très tôt essayé de programmer avec. Avant Internet, pour un collégien en province, c'était une galère pour avoir accès à de la doc, et surtout une méthodologie : je l'ai compris plus tard en arrivant à l'IUT Informatique en réapprenant l'algorithmie et en ayant accès (trop tard) à des vraies docs sur le Z80. Les bases de l'assembleur je les ai apprises par un ami qui était dans une filière électronique. Mais le vrai déclic fut quand j'ai eu accès à DAMS (pirate) et 2 autres amis qui

m'ont expliqué les bases : le balayage du rayon pendant la VBL, la théorie de la rupture (sans la pratiquer), etc. Je n'ai pas tout mis en pratique tout de suite, mais avec l'aide d'Amstrad 100% j'ai pas mal progressé.

**Tu te réintéresses depuis peu au CPC, quel a été l'élément déclencheur ?**

J'ai toujours eu des phases où je regardais tous les 3/4 ans ce qui se faisait sur CPC. Mais mon réintérêt pour le CPC s'est fait en plusieurs phases : j'étais tombé sur l'interview d'Antoine sur *Memory Full* et ça m'avait poussé à reprendre contact avec lui et le revoir juste avant le confinement. Je l'avais vu une dizaine d'années plus tôt dans le cadre d'activités universitaires et c'était agréable de le recontacter : Antoine a été important dans mon parcours sur CPC à la fois pour son amitié, mais aussi pour les échanges techniques.





Le gros déclic pour me réintéresser au CPC a eu lieu à l'été 2022. J'ai eu des démarches à faire et j'ai du fouiller dans tout mon bordel. J'ai remis la main sur la feuille de paramètres pour les musiques ST de Logon que Naminu (paix à son âme) m'avait données. Ça m'a pour le coup ramené à toute cette époque où je l'ai côtoyé. Je me suis mis en tête de voir ce que je pourrais ripper des mes anciennes disquettes. Et finalement j'ai rippé assez tôt la disquette des musiques ST d'époque (celle qui m'a servi pour *Time 2 Go* dans *Divine* par exemple), Rhaaaa Lovely comme dirait qui vous savez :). J'ai pu ripper pas mal d'autres choses, et j'ai retrouvé quelques messages électroniques (d'Antoine, Nicky One, Epsilon, par exemple) qui m'ont replongé dans ces années. J'ai rippé 25 % de ce que j'ai, mais des éléments essentiels pour moi ont été rippés.

**Ta période d'activité, de 1994 à 1996, a finalement été assez courte mais intense. Quels sont les meilleurs souvenirs que tu gardes de celle-ci ?**

En fait j'ai dû commencer en 1992 avant ma *Dragon Ball demo* et avant pour ma première demo *Victory Demo* sous le nom DCA (mes initiales et le titre est le dernier album des frères Jackson à l'époque). Ca fait 4 ou 5 ans d'activité tout de même. Les meilleurs souvenirs, ce sont les meetings (Swab 93, Bugs Meeting, 1Week2Stic, Freedelire, OVL 96), et l'esprit qui tournait à 1000 %, la sortie de nouvelles demos aussi et l'intégration dans des groupes.

Au sujet des groupes, j'ai finalement fait partie de Paradox (sans rien sortir), puis de Bugs (qui était sur la fin) avant de « former » des groupes juste parce qu'on avait des idées marrantes de nom : System SYSTEM puis Garbage Performers. Franchement je peux me retourner sur tout ça en me disant qu'on a beaucoup rigolé avec Barthy (show) et MaDe (Heuluile). On était insouciant, productifs peut-être insolents... Quelle époque fantastique que nous n'aurions pas vécue avec Internet. Parmi les meilleurs souvenirs, il y a forcément les rencontres avec Alexis Henaux, Pierre Leprun (Naminu), Antoine Pitrou (Antoine), Carlos Pardo (Made) et les Bugs.

**Tes scrollings étaient toujours remplis de déclarations lapidaires et polémiques qui ont marqué les esprits. Alors : tendance mégalo ou goût pour la provoc ?**

Difficile de placer le curseur entre les deux, et ça serait facile pour moi de tenter de redorer mon blason... Je ne pense pas être quelqu'un de prétentieux ni l'avoir été, mais j'ai un amour inné pour la chambrette. Et il faut se remettre dans le contexte ; je suis d'une génération plus jeune que les Logon d'origine. J'ai vu la scène s'étioier, avec de moins en moins d'innovations ou carrément des demos géniales qui ne sortaient pas par manque de motivation (par exemple la *Madness* de Gozeur s'est faite désirer, et la *5KB3* a eu un embargo). Il y avait peu de productions et pas mal de choses à la « Epsilon » : des demos correctes voire belles mais sans grosse innovation. Je pense que j'aurais été le premier content qu'un mec me cloue le bec avec une vraie innovation, même si clairement il y avait des choses très propres qui annonçaient la suite avec notamment l'arrivée d'Arkos (Rainbird si tu me lis : coucou).

Après il y a eu aussi beaucoup de chambrettes gentilles dans mes scrolls, notamment envers Barthy, Epsilon, etc. Les deux cités me l'ont d'ailleurs bien rendu (j'ai souvenir d'un logo « fro-Mage » par exemple) et franchement c'était marrant. Donc est-ce que j'ai été pré-

tentieux ou est-ce que c'était un moyen de susciter des réactions ? Chacun son avis et je les assume tous.

**Cela avait donné lieu à différentes bisbilles, dont une fameuse avec Chany et Grees. Comment résumerais-tu les choses aujourd'hui ?**

Si je devais résumer la brouille la plus connue, c'était avec le NPS. Je dirais que Chany a mis en fichiers ma demo *Dragon Ball Zeta* pour une raison louable : la copier sur des disquettes avec des secteurs défectueux. À l'époque on voulait optimiser les coûteuses disquettes vu qu'on était pour la plupart pas encore actifs, et mon trackloader obligeait à copier *Zeta* sur des disquettes sans aucun secteur défectueux. Le problème c'est que Chany a fait cette mise en fichiers sans m'en parler alors que ça s'est passé pendant / autour du Bugs Meeting 2 où on s'est côtoyés. Je l'ai mal pris et ça a basculé avec je l'avoue sûrement une sur-réaction de ma part envers ce mec qui revient à un chargement fichier et ajoute son nom au début. Je ne regrette pas, ça fait partie de mon passé et certaines phases m'ont amusé. Par la suite je crois qu'il y a eu des périodes contradictoires (au moins de ma part, je n'ai pas suivi tout ce qu'a fait Chany par la suite) : je me rappelle à la fois avoir prôné l'apaisement (de mémoire dans l'intro de la *Power System Megademo*), mais au final, mon dernier message (dans la part *Time 2 Go* de *Divine*) en remet une couche

avec une balle perdue pour AST. Je ne me rappelais plus pourquoi je m'en étais pris à AST alors que je m'étais bien entendu avec lui en meeting ; en repréparant cette interview, j'ai retrouvé un message, dans lequel je pensais qu'il était impliqué dans cette parodie de merde que je ne vais pas nommer. Plus de 20 après je me demande si on a su qui c'était.

J'ai regardé une partie de ce qu'a fait Chany depuis, il me reste une question : « Where is Chany ? » (j'espère que les lecteurs auront la ref).

**Tu étais assez proche d'Antoine à cette époque. Avec quels autres CPCistes étais-tu en contact ? Y avait-il beaucoup d'échanges techniques en privé ou aux meetings ?**

Parmi les belles rencontres, il y a Alexis Henaux (*AMS'ING Mag*, *Twilight Strad*, *Genesis*) que j'ai rencontré via *CPC Infos* n°30 alors qu'il habitait à 600m de chez moi. Ensemble on a beaucoup échangé CPC digits, etc. Puis on a vite dévié sur l'animation japonaise et les mangas, domaine dans lequel on a fait partie des précurseurs avec le fanzine *Kimagure Mangamag*. D'ailleurs à cette époque je suis donc dans « l'underground » avec le CPC, avec l'animation japonaise et manga avec *Kimagure Mangamag* et dans la foulée avec le Hip-Hop ; je mesure la chance d'avoir vécu tout ça. Il y a aussi les belles rencontres du Swab Meeting 93 : Steph, et MaDe (Carlos : Kuiiiiik !!), et finalement c'est lors de ce meeting que je commence à avoir des vrais contacts avec des membres de la scène. C'est sûrement dans la foulée que je rejoins les Bugs, dont Barthé avec qui on a plus que rigolé (« putain ce mec est coooooooooool »).

Je ne me rappelle plus trop le détail de la prise de contact avec Antoine. Il y a probablement eu une convergence depuis *DarkStrad*, pour qui j'avais fait une demo (qui ne doit pas être sortie), et qui m'a amené vers *Force One*, puis vers *MMPF* et Antoine. Je pense que ce qui a



« matché » avec Antoine, c'était sa façon d'écrire dans *MMPF* qui m'avait donné envie de le connaître. À l'époque où on fait connaissance, Antoine n'est pas encore le codeur ultra doué qui s'est révélé à partir de la *Subliminale / Shade Bobs* de la *Power System Megademo*, c'était un mec cool dont j'aimais le ton et c'était plus une amitié qu'une relation de co-deurs.

Dans les contacts importants il y a aussi eu CJC du CCC qui était un swapper important à l'époque et qui a connu pas mal de succès dans le domaine de la sécurité informatique. C'est le mec qui m'a appris à coller les timbres à la colle UHU (je laisse aux lecteurs la compréhension du pourquoi)... Epsilon a aussi été un ami à une époque, mais cette relation n'était pas équilibrée comme avec Antoine ou MaDe par exemple. Pour être positif à son sujet, je retiendrai qu'Epsilon (aka Xylophone) a été le ciment d'une grosse partie de la scène à un moment : il était partout et faisait tout (code, musique, gfx). J'aurais aimé qu'il se spécialise et innove plus par lui-même. Même s'il est (très) critiquable par certains aspects, la scène CPC lui doit beaucoup.

Un document qui m'a fait franchir un cap aussi est le document de Gozeur sur la rupture. Je pense que je l'ai reçu à la courte période où j'ai été membre de Paradox, et il contenait beaucoup d'informations utiles (en tout cas à l'époque quand on parlait de zéro, c'était utile, même si depuis on a fait mieux).

Ma plus grosse progression technique correspond à l'été 1995, durant lequel je suis en stage à Paris ainsi que Naminu (que j'ai rencontré via des amis de notre IUT commun à Dijon). Naminu m'a énormément appris : on se voyait au moins une fois par semaine pour la livraison des comics (#Dekoninck), et on parlait forcément énormément de CPC. C'est par exemple Naminu qui m'a parlé de la technique pour déplacer les bords des sprites en me parlant de la *5KB3* que j'ai utilisée dans *IpM Cult* sans avoir vu la *5KB3*. C'est encore Naminu

qui m'a filé les musiques transférées d'Atari ST par Fefesse, dont *A prehistoric tale* (de Jochen Hippel aka Mad Max) que j'ai utilisée dans *Time 2 Go* de *Divine*.

Les derniers échanges techniques ont eu lieu à la fin en privé avec Antoine notamment sur les specs de *Divine*. Et on parlait un peu de techniques autres, mais plutôt pour avancer sur cette demo : je me rend compte que j'étais un cancre sur tout ce qu'il essayait de me faire comprendre et il n'était que peu intéressé par le hard. Antoine a essayé de me faire arrêter le pré-calculé ou certains effets pas assez aboutis niveau Maths (y compris les vagues du haut pour lesquelles je n'avais clairement fait aucun effort d'originalité par rapport à Overflow dans *Best*). Antoine avait raison et je le savais, mais je crois que j'étais trop paresseux ou plus assez motivé.

**Peux-tu nous raconter la genèse de la *Divine Megademo* ? Certaines parties restent impressionnantes, mais elle a donné le sentiment d'être sortie dans un état presque inachevé...**

Les fichiers que j'ai retrouvés cet été m'ont amené plus de doutes que de certitudes sur la genèse de *Divine*. *A priori*, une demo commune entre Antoine et Odiesoft était prévue (sans moi), ainsi qu'une demo d'une face (wow!) de ma part pour laquelle j'avais invité Antoine sur une part. La demo *Shade Bobs* d'Antoine dans la *Power System Megademo* était d'ailleurs prévue, sous le nom *Subliminale* pour la *Mage Megademo*. On peut la retrouver sous ce nom sur certains sites d'ailleurs. À noter : quand je suis retombé sur *Subliminale* sur Internet à l'été 2022, je ne me rappelais plus de l'existence de cette version preview, que j'ai finalement retrouvée dans mes propres disquettes un peu plus tard.

Au final, pour *Divine*, je ne sais plus comment on a convergé sur 3 faces et la présence de Bollaware, mais il me reste quelques souvenirs que je vais lister en vrac :





- Antoine avait en projet de créer un filesystem compressé. Avec ses routines, il a créé les outils qui nous ont permis de mettre *Divine* sur disquette de façon assez pratique. Il me semble que c'est l'outil *S-DAF-Pro* que j'ai retrouvé mais pas encore retesté. Je me rappelle avoir pas mal échangé avec lui à ce sujet et des ajouts qu'il a bien voulu faire.

- on avait prévu d'avoir un loader à une adresse fixe, mais certaines parts ne reviennent pas au menu (je ne sais plus si c'est un bug ou des previews assemblées à la va-vite).

- le loader permettait de transmettre des paramètres entre les demos (langues, moniteur couleur, etc.), mais on ne l'a jamais exploité, idem pour un loader évolué qu'on avait envisagé. De mémoire, le loader gérait aussi les 2 faces du lecteur B, donc on pouvait mettre une face dans le lecteur A, les 2 autres sur le B et ne jamais avoir à changer de face. Hélas, je n'ai pas pu reproduire ce fonctionnement depuis l'époque.

- toujours en parlant de loader, j'ai retrouvé un embryon de turndisk par Antoine, c'est vraiment préalpha, et je comprends qu'on ne l'ait pas utilisé.

- j'ai découvert cet été (2022) sur *CPC Rulez* que MaDe avait fait des graphes pour mes parts dans *Divine* : un graphe *Time 2 Go* et un graphe *Regulate*. Le graphe *Regulate*, je suis

quasi-sûr que c'était pour moi car c'était le nom de travail de la part qui s'est finalement appelée *Dangerous* (faute d'avoir reçu le gfx *Regulate* ?). MaDe ne se rappelle plus non plus de ces deux graphes : on était tous en train de passer à autre chose.

- j'ai redécouvert cet été que *Coding Lesson* d'Antoine aurait facilement pu être intégrée dans *Divine* : j'ai une preview qui date de la période de développement de *Divine*, et j'ai retrouvé un message d'Antoine à propos des graphes à utiliser pour cette part, ce qui m'a permis de dater cette preview assez précisément.

- quand Naminu me parlait de la *5KB3*, j'avais été super frustré de ne pouvoir la voir ni l'avoir. À l'époque, les *5KB* avaient déjà quitté le CPC et il y avait une sorte d'embargo sur leur demo (qui ne venait pas de Naminu)... Quand l'énergie pour finir *Divine* n'était plus présente, il m'a semblé plus sympa de la sortir telle quelle en « advanced preview » plutôt que garder pour nous des parts qui étaient surprenantes, même en l'état.

- je me rappelle avoir eu finalement beaucoup de marge de manœuvre pour *Divine* : le choix du nom de la demo (coucou Hughes Grant), le choix du menu (dans l'absolu, j'ai honte du mauvais UX / Design, surtout qu'il manquait peu)... Je ne sais plus si j'ai décidé seul de la diffusion ou non, mais une chose est sûre : j'étais moteur pour ne pas que *Divine* dorme dans des cartons à jamais. La décision de diffusion s'est sûrement jouée entre mon début de service national et les études d'Antoine.

*Divine* est donc une « advanced preview » en quelque sorte, d'où le sentiment très légitime d'inachevé dont tu parles.

Au final, je crois que *Divine* a été un peu montrée en meeting avant sa diffusion (je préfère dire « diffusion » que « sortie » vu qu'elle n'a pas été finie), et je savais qu'elle susciterait une forte attente, avec la possibilité de déce-



voir. Sur certains aspects, ça n'a pas manqué : je sais que le menu qui revenait au début après chaque part a été beaucoup critiqué, et j'en prends l'entière responsabilité. Mais même sauvegarder l'emplacement du menu avant chaque demo était au dessus de ma motivation (alors que hyper facile techniquement).

Malgré tous les regrets d'avoir diffusé une « advanced preview », je reste plutôt fier de cette demo et de mes parties *IPM Cult*, *Dangerous* et surtout *Time 2 Go*.

Je n'ai compris que bien plus tard à quel point les parts d'Antoine et certaines parts d'Odie-soft étaient de vrais bijoux. Ca ne renforce que ma fierté d'avoir « posé » des parts dans la même megademo qu'eux ; fierté renforcée par quelques remarques que j'ai reçues à mon retour sur le net à cet été (2022).

**Quand as-tu réellement arrêté le CPC, et pour quelle(s) raison(s) ? As-tu poursuivi tes activités informatiques sur une autre machine par la suite ?**

En 1993, je découvre *Unix* et Internet en mode texte... Clairement la philosophie *Unix* est faite pour moi et depuis j'ai toujours un shell sous la main quel que soit l'OS sur lequel je travaille. Et puis quitter le CPC en 1996 c'était une certaine logique par rapport aux changements de générations : les anciens n'étaient plus très productifs, et les nouveaux pas encore. Donc je pense pas qu'on était les meilleurs, mais il y avait peu de monde avec qui nous pouvions discuter de façon équilibrée. Et puis en 1996, je commence une relation longue et je reprends la danse Hip-Hop, et en 1997 je reprends la fac en licence, ce qui m'amènera presque 10 ans plus tard à avoir des diplômes dont je n'aurais jamais rêvé ado.

La seule chose que j'ai faite en lien avec le demomaking est un affichage 3D fil de fer sous *Linux* en 1997 dans le cadre d'un projet à la fac. Il fallait mixer du C et de l'assembleur et plus dur encore : trouver la doc pour le faire.

La belle époque d'IRC avant mIRC, et des chanel #linuxfr #france2, etc. Et j'ai aussi suivi ce que faisait MaDe notamment avec Skaal, je ne me rappelle plus du nom de cette demo qui m'avait fait halluciné car Skaal avait reprogrammé un moteur 3D et un player MP3.

**Quel regard portes-tu sur la scène CPC actuelle, plus de 25 ans après ton départ ? Certaines productions t'ont-elles marqué plus que d'autres, et pour quelle(s) raison(s) ?**

La scène a fait un sacré bond, je suis clairement largué : techniquement il y a des choses auxquelles je n'aurais jamais pensé. Les demos sont souvent aussi beaucoup plus propres et homogènes en terme de design, c'est vraiment agréable. Je regrette toutefois l'absence de « vrais textes » dans les demos. Et ce côté très pro a aussi un revers : des demos trop « aseptisées » et ressemblant à la scène Amiga de l'époque, surtout sur le CPC Plus. On perd un charme spécifique au CPC, même si c'est bluffant.

Globalement, les productions qui m'ont le plus surpris sont :

- le compendium de Longshot et l'*Amazing Demo* (version 2021) : je pense qu'on a vraiment sous-estimé Longshot... Avoir quelqu'un d'aussi sérieux et méticuleux, c'est ce qui fait la différence de rendu et cohérence globale entre *Voyage 93* et *The Demo*. Merci Longshot #Revolog #Paris5
- les productions de No Recess : la version *Linux* de *CPCEC-GTK*, sa bibliothèque de DSK et SNA, mais surtout son port de *Sonic* qui a l'air vraiment impressionnant.
- la chaine *Youtube* et les astuces de Amaury / BDC Iron pour dumper les disquettes. Quand je suis reparti de rien, ça m'a vraiment beaucoup aidé, en plus du contact simple et facile avec un passionné ; notamment les commentaires sur des demos.

- la capacité de Chany a sortir des « demos » avec le même code en changeant juste la musique et le numéro dans le nom de la demo. Le seul truc plus dur à différencier sur terre, ce sont les chansons de Manu Chao. Impressionnant.

- je n'ai pas encore approfondi le sujet, mais certains outils comme *RASM* semblent vraiment sympas.

- toute la production d'Overflow. Pour moi c'est juste incroyable d'avoir innové autant à autant d'époques différentes. C'était un codeur hard de folie « à mon époque », il a encore pris une autre dimension avec le soft ces dernières années.

Ensuite il y a forcément des demos incroyables, mais je n'ai pas encore tout rattrapé et digéré. Parmi les incontournables, je vois *phX* de Condense, ou encore *Batman Forever* (meilleur storyboard de demo !), le travail des frères Rimauro. Désolé pour les travaux et demos que je n'ai pas encore vus... Je rattrape tout ça petit à petit.

### As-tu quelques projets en lien avec le CPC aujourd'hui ?

Je viens de finir de me rebricoler un « Ultimate CPC » : bouton hard reset, inverseur de disque B et A, sélection de face 2 sur lecteur externe, pause hard (juste pour le fun) mais surtout : interrupteur pour switcher entre mes deux CRTC 0 et 1A (un grand merci à Longshot pour l'idée, les photos et doc et l'aide pour ce faire). J'ai aussi un CPC avec Gotek interne, et un CPC (CRTC 2) avec lecteur d'origine et Gotek externe (merci à Rodrik). Ne me manquent qu'un CPC Plus et peut-être un 664 :)

Pour le futur... en toute honnêteté, j'ai eu en tête de coder la demo que je n'avais pas commencée à l'époque, mais je me suis rendu compte que je suis complètement rouillé. Je n'arrive d'ailleurs pas encore à recompiler mes anciens sources...

Seul « mini projet » : diffuser quelques previews de demos sorties que je compte poster en mars 2023 juste pour le symbole (mes 50 ans), ça sera forcément plus symbolique que spectaculaire.

### Merci de m'avoir accordé un peu de ton temps, je te laisse le dernier mot !

Jean-Pierre (c'est mon dernier mot... Désolé).

Plus sérieusement, un grand bravo à tous ceux qui sont devenus ou restés actifs depuis 1995. C'est vraiment impressionnant de voir des choses différentes :

- comment le CPC a été poussé dans des choses qu'on aurait jamais pensé réalisables ;
- l'entraide et la passion de certains sur les sites de Retro Gaming.
- le travail d'inventaire complètement hallucinant réalisé par certains sites.

Quelques incontournables en 2022 : *Genesis 8bit* (longévité et sérieux incroyables), *CPC-Power*, *Amstrad CPC Mémoire Ecrite*, *Memory Full*, et enfin *CPC Ruez* pour les sites consacrés uniquement au CPC.

À propos de pérennisation, j'espère que tout le recensement de ce patrimoine ne disparaîtra pas avec leurs auteurs... Je rêve d'un monde où ils pourraient mettre à disposition de tous un dump de leur travail sans être pillés et non cités.

Merci de m'avoir lu.

« Imperial Mage » aka « j'me rappelle plus de mon nom complet »



# 64 NOPs

**Rédaction :** Hicks, toms

**Ont contribué à ce numéro :** CNGSoft, Eliot,  
Golem13, Madram, Mage, OffseT, Roudoudou, Zik

**Couverture :** Morgane BASSOMPIERRE SEWRIN  
[www.instagram.com/little\\_blue\\_owl\\_/](https://www.instagram.com/little_blue_owl_/)

**Mise en page :** toms

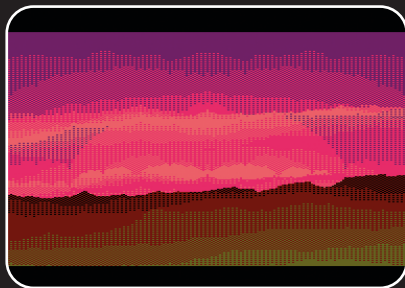
**Pour contacter la rédaction :**

**Email :** [admin@memoryfull.net](mailto:admin@memoryfull.net)

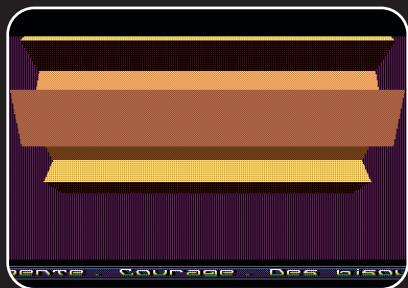
**Sites Web :** [64nops.wordpress.com](http://64nops.wordpress.com) · [www.memoryfull.net](http://www.memoryfull.net)



*Onescreen Colonies 2 (Vanity)*



*Open Space (Vanity)*



*Debris (Pulpo Corrosivo)*



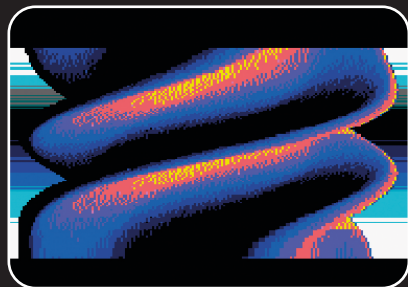
*Veteran Megademo (Impact)*



*Foursome (Pulpo Corrosivo)*



*Sphere (Resistance)*



*FMR CPC (SMFX)*