LE MAGAZINE DES PROGRAMMEURS SUR AMIGA - N°26 OCT. 1991

AMIGA NEWS-TECH

EN VENTE UNIQUEMENT PAR ABONNEMENT

LANGAGE
ARexx : travail pratique (I), par F.G. (p.6)
Amos : trucs en vrac (II), par F.L. (p.4) et Daisy Derata (p.5)

EXECBASE
ExecBase I : déboguer avec Monam2, par D.J. (p.2)
ExecBase II : les interruptions sous Exec, par S.S. (p.22)

TOOLBOX
ToolBox : Draco, par H.D.V (p.8)

SUB.way
SUB.way I : les Layers, par M. (p.18)
SUB.way II : bougez avec intuition et graphics (II), par H.D.V (p.25)

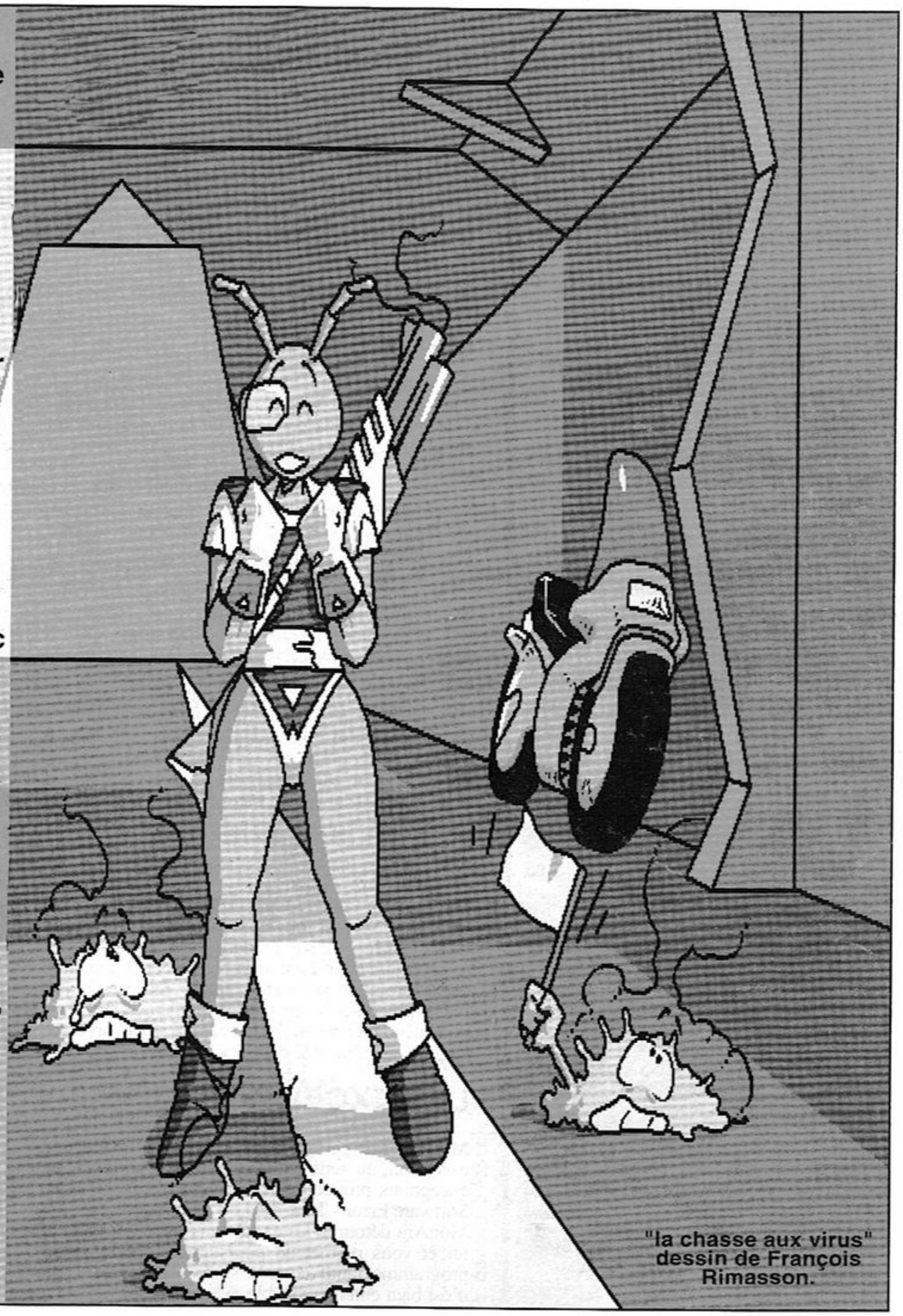
DEMO-MAKERS
Le Trial Playfield! par J.E. (p.10)

TRANSACTOR
les rouleaux de lumière, par F.B (p.28)

HARDWARE
Touche pas à mon clavier, par L.F. (p.20)

UTILITAIRE
NewPic, par M. (p.12)

REQUESTER 30



"la chasse aux virus" dessin de François Rimasson.

45 F.

Ce numéro 26 de l'Amiga News-Tech a été tiré à 3500 exemplaires.
Ce magazine est une publication Commodore Revue Sarl. 5. rue de la Fidélité - 75010 PARIS.

EDITO

Tragédie à la rédaction... Yves Huitric, notre bien aimé Rédacteur en Chef, nous à quitté. Il a deux années durant oeuvré pour vous offrir l'Amiga Revue et l'ANT qui vous sont si chers (30 F !). Mais ne le pleurez pas, car mon petit doigt me dit que nous le retrouverons très bientôt dans l'univers de l'AmigaDos. Une aventure à suivre...

Plus de CATS ! C'est terminé, cette rubrique a vécu. Désolé pour ceux qui attendaient la suite de l'article de Léo Schwab (peu nombreux d'après vos réactions...). CATS a vécu, toutes les recommendations officielles du Commodore Amiga Technical Support y sont passées. Vous trouverez dès le prochain numéro une nouvelle rubrique pour la remplacer. Pour l'heure, nous en avons profité pour nous faire un peu d'auto-publicité, ça ne fait jamais de mal par où ça passe.

Autre grande nouveauté dans l'ANT : il y a dorénavant un coupon d'abonnement à découper ou/et à photocopier, si d'aventure vos petits copains avaient envie de faire partie de notre grande famille. Après six numéros, il était temps !
D'ailleurs, en parlant de ça, nous sommes en train de mettre au point une formule de parrainage des abonnements, je ne vous dis que ça.

Les RKM en français ? Ca vient. Il ne s'agira peut-être par d'une véritable traduction des RKM originaux, droits de copyright oblige, mais de livres écrits par les piliers de la rédaction. A commencer par le doyen, j'ai nommé Tonton Max, qui bosse actuellement comme un fou sur ce qu'il connaît le mieux : l'assembleur.

Stéphane Schreiber

OURS

Je commence à en avoir marre : ça fait six mois qu'on vous le répète inlassablement, et au train où c'est parti, ça va encore durer longtemps. D'abord, qui c'est qui la le lire, cet ours, hein, qui? A part le Big Boss parce qu'il a peur que je profite des petits caractères pour me foutre de sa gueule.

Bon, allez, une fois de plus, sacrifions à la tradition : Amiga NewsTech est une publication de **Commodore Revue SARL, société sise à Paris dans le 10ème arrondisse- ment, au 5, rue de la Fidélité.**

Le Directeur de la Publication, c'est Jean-Yves Primas. Mensurations : **(1) 42.47.12.16** au repos, sinon, je suis pas. Signes particuliers : Adore les douaniers italiens.

Le nouveau rédacteur en chef, c'est bibi, mais chut, le dites pas à ma mère, elle croit que je suis pompier.

Toute la maquette est faite en PAO sur Amiga (comme quoi on n'a vraiment peur de rien). Professional Page 2.0 me fait doucement rigoler, mais on n'a que ça sous la main. Misère.

Les pigistes sont (par ordre d'apparition à l'écran) : Bibi, Denis Jarril, François Lionet, François Gueugnon, Pascal Amiable, Jerôme Etienne, Max, Stéphane Schreiber, Loïc Far, Herr Dolor, François Brion (le Transactor du mois), Ok et Cancel (le Requester) et Madame Maria (la femme de ménach').

La vente par correspondance et les problèmes d'abonnements sont gérés par la société **MCM Europe, 16, quai Jean- Baptiste Clément, 64150 AlfortVille. Tel : (1) 43.78.92.10.**

Flashage : ColorWay, 5, rue de la Fidélité, 75010 Paris. Tel : (1) 40.78.07.87.

Impression : N.L.C., Reims. Tel : (16) 26.07.57.87.

ExecBase : DEBOGUER

Après une longue discution téléphonique avec l'un de nos innombrables abonnés, je me suis rendu compte que peu de gens savaient utiliser efficacement ce superbe et indispensable outil qu'est MonAm.

Pourtant, ledit abonné était un utilisateur "officiel" du Devpac, c'est-à-dire qu'il l'avait acheté tout-à-fait légalement, et était même allé jusqu'à renvoyer sa carte de garantie à HiSoft (ce que, malheureusement pour eux, peu de Français font). C'est à ces gens-là uniquement que cet article s'adresse ; il ne servira absolument de mode d'emploi aux utilisateurs "officieux", qui se seraient procuré le Devpac par des voies détournées...

POURQUOI UN DEBOGUEUR ?

C'est bien connu, les programmes écrits en langage machine sont plus particulièrement sujets à des erreurs. Celà peut aller de la plus bénine (la simple erreur d'affichage, très simple à corriger, qui quit entraîne une illibilité totale du texte), jusqu'à la sérieuse (résultat d'un calcul erroné) et à la carrément catastrophique (plantage de l'Amiga). Un débogueur vous aide à trouver ces erreurs et à les corriger. Il se présente généralement sous la forme d'un désassembleur qui vous permet d'exécuter votre programme pas-à-pas, instruction par instruction. Vous pouvez ainsi facilement repérer les endroits de votre programme qui ne réagissent pas comme vous l'auriez souhaité.

MonAm est un débogueur symbolique, c'est-à-dire qu'il désassemble le code machine en utilisant tous les labels définis dans votre programme source, exactement comme si vous vous trouviez encore dans l'éditeur, GenAm2. Cette présentation est évidemment d'une aide très précieuse et simplifie grandement la recherche des bogues. La seule condition à ceci est que le programme exécutable contienne un (ou plusieurs) HUNK_SYMBOL, ce que GenIm2, l'assembleur du Devpac, sait parfaitement faire (assembler avec l'option 'Debug Info' sur 'Normal' ou 'Exports'). Notez enfin que bien que MonAm soit un débogueur de bas-niveau (c'est-à-dire ne présentant que les codes machine du programme et les registres du micro-processeur), on peut également s'en servir pour déboguer des programmes écrits en un autre langage, comme le C ou le Pascal.

LES EXCEPTIONS

MonAm sait parfaitement reconnaître quand une exception, au sens 68000 du terme, survient. Ces exceptions provoquent normalement un requester Software Error - Task held ou un Guru Meditation. MonAm détourne les exceptions sur une routine à lui et vous permet de poursuivre l'exécution du programme fautif comme si de rien n'était. Celà n'est bien entendu valable que pour le programme en cours de débogage ; si une autre tâche venait à provoquer une exception, le Guru apparaîtrait de toute manière.

Pour tracer un programme instruction par instruction, MonAm utilise ce que l'on appelle des points d'arrêt. Pour placer un tel point d'arrêt dans un programme, MonAm y insère le code machine S4AFC, correspondant à l'instruction 68000 ILLEGAL. Son convertisseur d'exception intégré est alors capable de déterminer si cette instruction définissait un point d'arrêt, ou si elle était le fruit d'une erreur dans le programme. Dans les deux cas, MonAm suspend l'exécution du programme, mais la suite des évènements dépend bien sûr de la raison de la présence de l'instruction ILLEGAL ici. Nous reviendrons plus en détails sur les points d'arrêts un peu plus loin.

L'ECRAN DE MONAM

L'illustration quelque part dans cette page, représente l'écran de MonAm, avec toutes les fenêtres qu'il est possible d'y ouvrir (5 en tout). La première présente le les registres du 68000 (leur contenu et quelques octets de la mémoire qu'ils pointent) ; la seconde est la fenêtre de désassemblage principale ; la troisième présente un dump hexadécimal et ASCII de la mémoire ; la quatrième fenêtre peut être utilisée soit pour désassembler une autre portion du programme que celle affichée dans la fenêtre 2, soit pour visionner un texte source, comme ici ; enfin, la cinquième fenêtre présente un second dump mémoire, très pratique pour comparer une zone avec celle affichée dans la fenêtre 3.

En mode trace, MonAm rafraichit le contenu de toutes ses fenêtres après chaque instruction exécutée. Celà vous permet de suivre en temps réel la modification d'un tableau en mémoire, par exemple. De plus, et à l'exception de la première, toutes les fenêtres de MonAm peuvent être liées à un registre particulier. Ainsi, après chaque instruction, l'adresse de la mémoire représentée par cette fenêtre est recalculée. Par défaut, la fenêtre 2 est liée au PC.

LES EXPRESSIONS NUMERIQUES

MonAm dispose d'un véritable interprêteur pour évaluer les expressions que vous lui fournissez soit directement, soit en tant que paramètre d'une commande. Cet interprêteur prend même en compte la priorité des opérateurs arithmétiques !

MonAm considère que tous les nombres entrés sont en hexadécimal. Pour entrer un nombre en décimal, il faut le précéder du signe \. Quand un nombre hexadécimal pourrait être confondu avec un nom de registre, il faut le faire précéder du signe $ ou d'un 0. Vous pouvez également utiliser les registres du 68000 dans une expression. Quant aux accolades { et }, elle représentent le contenu de l'adresse pointée par l'expression qu'elles entourent. Ainsi, des expressions comme

```
d4*\12+{a0}
{mon_tableau+d0}.w
d0&~d1
```

sont parfaitement valides.

AVEC MONAM2

Il existe également plusieurs symboles réservés :

```
CODE    Adresse du premier hunk du programme
HUNKx   Adresse du xème hunk du programme
SP  Pointeur de pile courant (USP ou SSP)
SSP Pointeur de pile superviseur
SR  Registre d'état du processeur
```

Enfin, 10 variables, numérotées de M0 à M9 peuvent prendre les valeurs de votre choix. Par défaut, les variables M2 à M5 contiennent l'adresse de début des fenêtres correspondantes. M0 et M1 sont positionnés, lorsque l'on charge un fichier binaire (pas un programme !) en mémoire, respectivement sur l'adresse de début et l'adresse de fin du fichier.

LES POINTS D'ARRET

Les points d'arrêt de MonAm sont conditionnels, c'est-à-dire que l'on peut préciser certaines conditions que MonAm évaluera avant de décider s'il doit interrompre ou non l'exécution du programme en cours.

La première forme de point d'arrêt est la plus simple : le programme s'interrompra une fois ce point d'arrêt atteint, et le point d'arrêt sera effacé. L'utilisation la plus évidente est de permettre l'exécution du programme jusqu'à ce qu'une routine particulière soit atteinte.

La seconde forme de point d'arrêt est permanente ; chaque fois que le programme passera sur ce point d'arrêt, il sera interrompu.

La troisième forme de point d'arrêt introduit la notion de compteur ; le programme ne s'arrêtera qu'après être passé un certain nombre de fois (défini par vous) sur ce point d'arrêt, ce qui est très utile dans les boucles.

Enfin, la forme la plus puissante est le point d'arrêt conditionnel, c'est-à-dire que le programme ne sera interrompu que si la condition spécifiée est remplie. Cette condition peut aussi bien être un registre qui prend une valeur particulière, qu'une zone de mémoire qui soit modifiée, soit encore que le bit Z de SR soit positionné. Toutes les expressions vues plus hauts sont valides.

La commande Amiga droite-B, qui permet de placer un point d'arrêt à une adresse donnée, accepte plusieurs formes de paramètres :

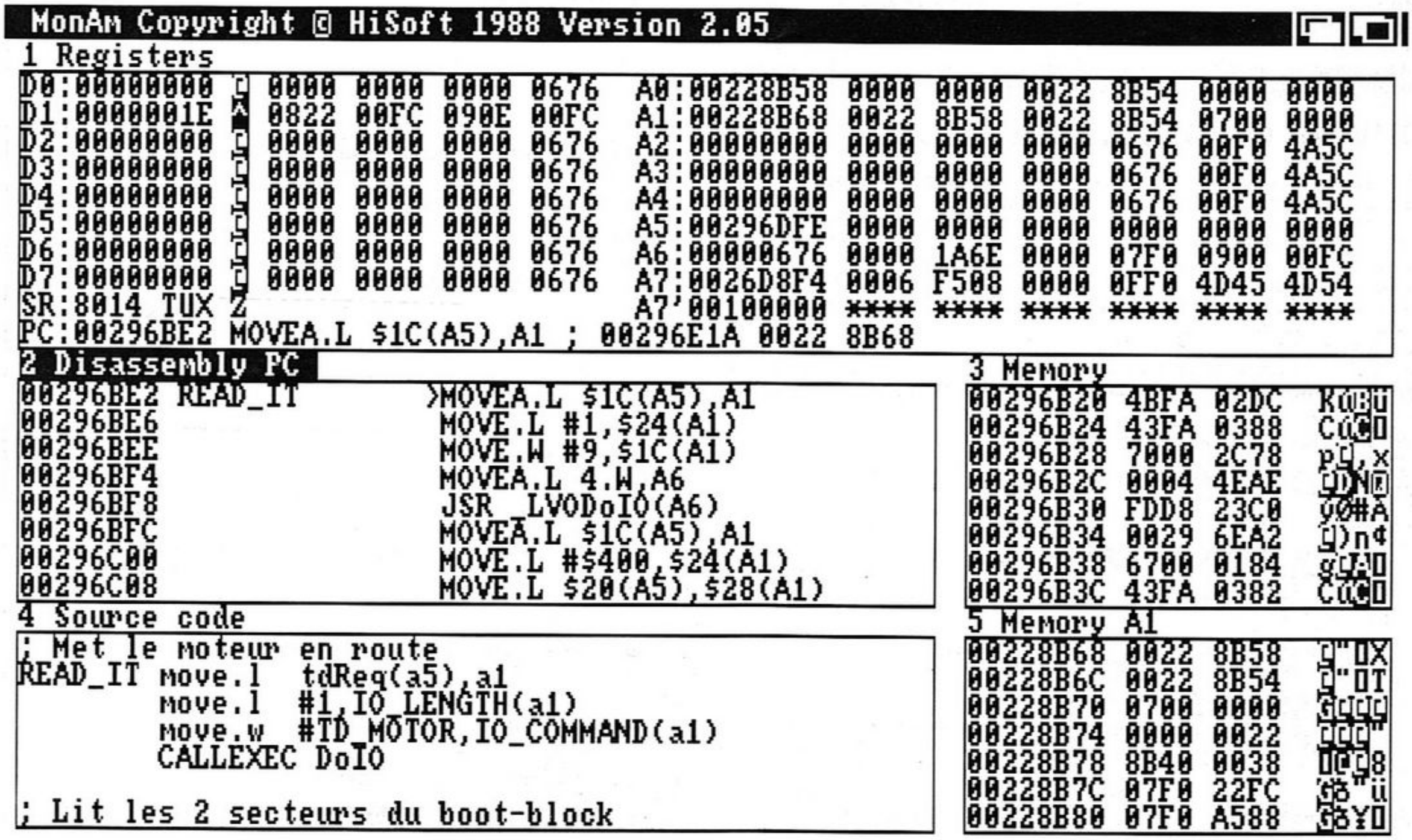
```
<adr>   place un point d'arrêt simple
<adr>,<exp> place un point d'arrêt à <exp> passages
<adr>,* place un point d'arrêt permanent
<adr>,?<exp>    place un point d'arrêt conditionnel
<adr>,- supprime un point d'arrêt
```

Dans la table ci-dessus, <adr> représente une adresse quelconque (par exemple, un label du programme), et <exp> une expression valide. MonAm ne peut pas placer de points d'arrêt à des adresses impaires ni en Rom.

MONAM CONTRE ADEBUG

Voilà pour l'essentiel de MonAm. Nous continuerons le mois prochain notre survol de ses principales fonctionnalités, en même temps que vous pourrez trouver dans l'ANT le test d'un nouveau débogueur, français celui-là, j'ai nommé ADebug.

Par Denis Jarril



AMOS : TRUCS EN VRAC (II)

Un peu de tout aujourd'hui. De la programmation système pour commancer, puis une petite Daisy_DemOS. Ma célèbre chienne vient juste de terminer le compilateur et est un peu fatiguée, la pôvre.

J'aimerais me poser une question : "Dis donc François, pourquoi que tu n'as pas mis de fonction pour récupérer la date et l'heure du système en AMOS ?". Hé bien, c'est une bonne question, mon poil dans la main doit certainement connaître la véritable raison. Quoi qu'il en soit, je vous propose deux procédures à insérer dans vos programmes, effectuant le boulot.

La première procédure, astucieusement appelée _DATE$, vous retourne, devinez quoi, j'en vois un qui n'écoute pas dans le fond, la date, bravo. Elle attend un paramètre en entrée : 0 pour avoir la date en format jj-mm-aaaa et tout nombre différent de 0 pour l'avoir sous forme littérale (par exemple, Mardi 11 Juin 1991).
La deuxième procédure, non-moins astucieusement appelée _TIME$, vous retourne l'heure (et non pas un paire de claques comme celle que le petit Stephane S. assis au fond vient de glisser à son voisin).

Quelques remarques subtiles qui me passent par la tête :
- gardez le souligné ("_") avant le nom de ces procédures, il n'est pas impossible que je force Daisy à programmer des versions langage machine de ces procédures, auquel cas les mots-clés DateS et Time$ deviendront réservés.
- j'appelle la fonction DOS DateStamp(), qui retourne des valeurs tordues (nombre de jours écoulés depuis le premier Janvier 1978 pour la date, et nombre de minutes depuis minuit, ainsi que le nombre de 1/50iemes de seconde depuis le début de la minute pour l'heure). Pour une fois, on peut dire que les programmeurs du système ne se sont pas trop fatigués.
- la méthode de calcul de la date n'est pas des plus rapide, mais je n'avais pas envie de vous faire taper des milliers de datas. A propos de datas, notez la taille du mois de Février : AMOS permet de faire des calculs dans les datas, ce qui est très pratique ici !

OUVRIR UN DEVICE EN AMOS

Contrairement à ce que disent les mauvaises langues jalouses, AMOS respecte farpaitement le système. Il est tout-à-fait possible d'ouvrir un device en basic, comme le prouve notre deuxième programme du jour.
Nous ouvrons ici le trackdisk.device, qui permet d'accéder piste par piste à une disquette. Le programme ne fait pas grand chose, à part ouvrir ledit device. On ressent néanmoins l'immense satisfaction d'avoir ouvert un tel device en AMOS. J'adresse sollenellemnt 156831,95 mercis à Nicolas Costes de Commodore France - pardon, de l'Armée Française maintenant... - pour m'avoir aidé...
Le mois prochain, nous réaliserons une routine d'évaluation d'une expression quelconque, très pratique pour réaliser des traceurs de courbe.

```
'------------------------------
' Lecture date et heure en AMOS
'------------------------------
_DATE$[1] : Print Param$
_TIME$ : Print Param$
'
Procedure _DATE$[TYPE]
    '
    ' Appel de la fonction DOS: DateStamp
    T$=Space$(12)
    Dreg(1)=Varptr(T$)
    RIEN=Doscall(-192)
    NJ=Leek(Varptr(T$))
    '
    ' Trouve l'année et le premier jour de l'année
    A=1978 : JOUR=7
    Do
        BIS=0 : If(A and 3)=0 : BIS=1 : End If
        Exit If NJ-365-BIS<0
        Add JOUR,1+BIS : If JOUR>7 : Add JOUR,-7 : End If
        Add NJ,-365-BIS
        Inc A
    Loop
    '
    ' Trouve le mois
    M=1
    Do
        Read N
        Exit If NJ-N<0
        Add NJ,-N : Inc M
    Loop
    Inc NJ
    '
    ' Fabrique la chaine
    If TYPE=0
        J$=Mid$(Str$(NJ),2) : If Len(J$)<2 : J$="0"+J$ : End If
        M$=Mid$(Str$(M),2) : If Len(M$)<2 : M$="0"+M$ : End If
        A$=Mid$(Str$(A),2)
        DATE$=J$+"-"+M$+"-"+A$
    Else
        Restore JOURS
        For N=1 To JOUR
            Read J$
        Next
        Restore MOIS
        For N=1 To M
            Read M$
        Next
        DATE$=J$+Str$(NJ)+" "+M$+Str$(A)
    End If
    '
    ' Longueur des mois de l'annee
    Data 31,28+BIS,31,30,31,30,31,31,30,31,30,31
    '
    ' Noms des jours
    JOURS:
    Data
"Lundi","Mardi","Mercredi","Jeudi","Vendredi","Samedi","Dimanche"
    '
    ' Noms des mois
    MOIS:
    Data "Janvier","Fevrier","Mars","Avril","Mai","Juin"
    Data "Juillet","Aout","Septembre","Octobre","Novembre","Décembre"
    '
End Proc[DATE$]
Procedure _TIME$
    '
    ' Appel de la fonction DOS: DateStamp
    T$=Space$(12)
    Dreg(1)=Varptr(T$)
    RIEN=Doscall(-192)
    MN=Leek(Varptr(T$)+4)
    SEC=Leek(Varptr(T$)+8)
    '
    ' Calcul des minutes
    H=MN/60 : H$=Mid$(Str$(H),2) : If Len(H$)<2 : H$="0"+H$ : End If
    M=MN mod 60 : M$=Mid$(Str$(M),2) : If Len(M$)<2 : M$="0"+M$ : End
If
    '
    ' Calcul des secondes
    S=SEC/50 : S$=Mid$(Str$(S),2) : If Len(S$)<2 : S$="0"+S$ : End If
    '
    ' Fabrication de l'heure
    TIME$=H$+":"+M$+":"+S$
    '
End Proc[TIME$]
```

```
'----------------------------------------
' Ouverture du trackdisk.device en AMOS
'----------------------------------------
'
L=1024 : Gosub INIT_STRUCTURES
'
_CREATE_PORT["",0] : _PORT=Param
'
If _PORT
    _CREATEIO[_PORT,82] : _IO=Param
    '
    If _IO
        _OPEN_DEVICE["trackdisk.device",0,_IO,0]
        '
        If Param=0
            Print "Device ouvert, pressez une touche!"
            Wait Key
            _CLOSE_DEVICE[_IO]
        End If
        '
        _DELETEIO[_IO]
    End If
    '
    _DELETE_PORT[_PORT]
End If
'
End
'
INIT_STRUCTURES:
'
Global STRUCTURES
Erase 15 : Reserve As Work 15,L
Fill Start(15) To Start(15)+L,0
STRUCTURES=Start(15)
'
' Library offsets
'
Global _LVOOPENDEVICE,_LVOCLOSEDEVICE
Global _LVOADDPORT,_LVOREMPORT
Global _LVOFINDTASK
Global _LVOALLOCSIGNAL,_LVOFREESIGNAL
Global _LVODOIO
_LVOOPENDEVICE=-444 : _LVOCLOSEDEVICE=-450
_LVOADDPORT=-354 : _LVOREMPORT=-360
```

```
_LVOFINDTASK=-294
_LVOALLOCSIGNAL=-330 : _LVOFREESIGNAL=-336
_LVODOIO=-456
'
' Node definitions
'
Global _LN_NAME,_LN_PRI,_LN_TYPE
_LN_NAME=$A : _LN_PRI=$9 : _LN_TYPE=$8
'
' Message port
'
Global NT_MSGPORT,MP_MSGLIST,MP_SIGTASK,MP_SIGBIT,MP_FLAGS
Global PA_SIGNAL
NT_MSGPORT=$4  :  MP_MSGLIST=$14  :  MP_SIGTASK=$10  :  MP_SIGBIT=$F  :
MP_FLAGS=$E
PA_SIGNAL=$0
'
' IO
'
Global MN_REPLYPORT,MN_LENGTH,NT_MESSAGE
Global IO_COMMAND,IO_OFFSET,IO_LENGTH,IO_DATA
MN_REPLYPORT=$E : MN_LENGTH=$12 : NT_MESSAGE=$5
IO_COMMAND=28 : IO_OFFSET=44 : IO_LENGTH=36 : IO_DATA=40
'
' Device commands
'
Global CMD_READ,CMD_WRITE
CMD_READ=2 : CMD_WRITE=3
'
Return
'
Procedure _DOIOR[REQ,CMD,OFS,L,D]
    Doke REQ+IO_COMMAND,CMD
    Loke REQ+IO_OFFSET,OFS
    Loke REQ+IO_LENGTH,L
    Loke REQ+IO_DATA,D
    Areg(1)=REQ
    F=Execall(_LVODOIO)
End Proc[F]
Procedure _OPEN_DEVICE[NAME$,UNIT,IO,FLAGS]
    NPOKE[NAME$,0] : Areg(0)=Param
    Dreg(0)=UNIT
    Areg(1)=IO
    Dreg(1)=FLAGS
    F=Execall(_LVOOPENDEVICE)
End Proc[F]
Procedure _CREATE_PORT[NAME$,PRI]
    '
    Dreg(0)=-1 : SIGBIT=Execall(_LVOALLOCSIGNAL)
    '
    If SIGBIT<>-1
        '
        PRT=STRUCTURES : Add STRUCTURES,34
        '
        NPOKE[NAME$,PRT+10]
        Poke PRT+_LN_TYPE,NT_MSGPORT
        Poke PRT+_LN_PRI,PRI
        '
        _FIND_THIS_TASK : Loke PRT+MP_SIGTASK,Param
        Poke PRT+MP_SIGBIT,SIGBIT
        Poke PRT+MP_FLAGS,PA_SIGNAL
        '
        If Leek(PRT+_LN_NAME)
            Areg(1)=PRT
            F=Execall(_LVOADDPORT)
        Else
            _NEWLIST[PRT+MP_MSGLIST]
        End If
        '
    End If
    '
End Proc[PRT]
Procedure _CREATEIO[PRT,S]
    IO=STRUCTURE : Add STRUCTURE,S
    Poke IO+_LN_TYPE,NT_MESSAGE
    Doke IO+MN_LENGTH,S
    Loke IO+MN_REPLYPORT,PRT
End Proc
Procedure _DELETEIO[PRT]
    Poke IO+_LN_TYPE,-1
End Proc
Procedure _DELETE_PORT[PRT]
    '
    If Leek(PRT+_LN_NAME)
        Areg(1)=PRT : F=Execall(_LVOREMPORT)
    End If
    '
    Loke PRT+MP_SIGTASK,Leek(PRT+MP_SIGTASK)-1
    Dreg(0)=PRT+MP_SIGBIT : F=Execall(_LVOFREESIGNAL)
    '
End Proc
Procedure _CLOSE_DEVICE[IO]
    Areg(1)=IO
    F=Execall(_LVOCLOSEDEVICE)
End Proc[F]
Procedure _NEWLIST[P]
    Loke P+4,0
    Loke P+8,P
    Loke P,P+4
End Proc
Procedure NPOKE[N$,A]
    '
    If Len(N$)=0 and A<>0
        Loke A,0
    Else
        If A
            Loke A,STRUCTURES
        Else
            A=STRUCTURES
        End If
        '
        For N=1 To Len(N$)
            Poke STRUCTURES,Asc(Mid$(N$,N,1)) : Inc STRUCTURES
        Next
        Poke STRUCTURES,0 : Inc STRUCTURES
        '
        STRUCTURES=(STRUCTURES+1) and $FFFFFFFE
    End If
    '
End Proc[A]
Procedure _FIND_THIS_TASK
    Areg(1)=0 : F=Execall(_LVOFINDTASK)
End Proc[F]
```

Par François Lionnet

DAISY-DERATA

Vous avez remarqué comme il est gentil aujourd'hui ? Je ne sais pas ce qu'il lui prend, mais c'est louche. Il va certainement me demander de programmer quelque chose. Avec lui c'est toutou-rien !
Bien, voilà une p'tiote démo qui montre comment astucieusement réaliser d'énôôôrmes vu-mètres, gérés en AMAL. Vous pouvez donc parfaitement afficher un scrolling de texte défilant par dessus, sans les perturber le moins du monde.

```
'-----------------
' Daisy-DemOS 3
' Par Daisy Lionet
'-----------------
'
' Charger une banque de musique!
'
XBASE=128 : YBASE=50 : SX=160 : SY=100
'
Hide On
'
For S=0 To 3
    Screen Open S,SX*3,SY*2,4,Lowres
    Curs Off : Flash Off : Cls 0
    Palette 0,$FF0,$F42,$4A,,,,,0,$FF0,$F42,$4A
Next
'
Screen Display 0,XBASE,YBASE,320,SY
Screen Display 1,XBASE,YBASE,320,SY
Screen Display 2,XBASE,YBASE+SY,320,SY
Screen Display 3,XBASE,YBASE+SY,320,SY
'
Wait Vbl
'
Dual Playfield 0,1
Dual Playfield 2,3
'
Screen 0 : Ink 1 : Bar 0,0 To SX,SY
Screen 1 : Ink 1 : Bar SX*2,0 To SX*3,SY
Screen 2 : Ink 1 : Bar 0,SY To SX,SY*2
Screen 3 : Ink 1 : Bar SX*2,SY To SX*3,SY*2
'
Channel 0 To Screen Offset 0
Channel 1 To Screen Offset 1
Channel 2 To Screen Offset 2
Channel 3 To Screen Offset 3
'
A$=A$+"      Let R0=63;"
A$=A$+"Loop: Pause; Let R1=Vu("
'
B$=B$+");"
B$=B$+"      If R1=0 Jump Ret;"
B$=B$+"      Let R0=R1; Jump Set;"
B$=B$+"Ret:  If R0<1 Jump Loop;"
B$=B$+"      Let R0=R0-RA;"
B$=B$+"      If R0>0 Jump Set;"
B$=B$+"      Let R0=0;"
B$=B$+"Set:  Let R2=R0*2; Let R3=R0*3/2;"
'
C$="Let X=R2/2*2+1; Let Y=R3+1; Jump Loop"
Amal 0,A$+"0"+B$+C$
'
C$="Let X="+Str$(SX)+"-R2/2*2-1; Let Y=R3+1; Jump Loop"
Amal 1,A$+"1"+B$+C$
'
C$="Let X=R2/2*2+1; Let Y="+Str$(SY)+"-R3; Jump Loop"
Amal 2,A$+"2"+B$+C$
'
C$="Let X="+Str$(SX)+"-R2/2*2-1; Let Y="+Str$(SY)+"-R3; Jump Loop"
Amal 3,A$+"3"+B$+C$
'
Amreg(0)=2
Amal On
'
Music 1 : Led Off
'
Wait Key
```

Par Daisy Lionnet

AREXX : TRAVAIL PRATIQUE (I)

Comme nous l'avions promis précédemment, nous allons maintenant réaliser et analyser un programme ARexx. Plusieurs concepts nouveaux d'ARexx y seront abordés, que nous présenterons dans le détail le moment venu.

Le programme que nous proposons est une application d'ARexx en tant que langage de programmation (c'est-à-dire en laissant momentanément de côté ses qualités de "langage du multitâche"), ce qui est l'une de ses possibilités. Nous allons donc réaliser un programme qui extrait le répertoire d'un disque dur ou souple et le présente de façon plus agréable que celle du Workbench standard.
Le programme est divisé en trois modules fonctionnels indépendants qui s'interfacent les uns avec les autres et permettent, pourvu que l'on respecte cet interfaçage, d'en modifier un sans avoir à ré-écrire les autres. Ces trois modules se chargent respectivement de la prise en compte de la demande de l'opérateur, de l'analyse du contenu du disque sélectionné et de la mise en forme des informations obtenues précédemment.

Nous n'analyserons aujourd'hui que la partie "prise en compte de la demande de l'opérateur" ; le reste viendra après.

AVERTISSEMENT

Nous avons cherché à utiliser le maximum de fonctions ARexx sans nous préoccuper d'une solution astucieuse. Ce qui signifie que nous ne soutiendrons pas de polémique sur le bien-fondé de telle ou telle méthode. Cependant et à titre d'exercice, il n'est pas interdit de réfléchir à d'autres solutions...

DEFINITION DU MODULE D'ACQUISITION

Ce module permet de détrerminer le disque choisi par l'opérateur et d'acquérir certaines informations relatives à ce disque. Trois types d'opérateurs ont été envisagés : le vétéran, qui sait ce qu'il veut et ne se trompe pas dans l'appel ; l'indécis, qui ne sait pas ce qu'il veut ; et le novice, qui se trompe dans sa demande.
Pour éviter la typographie, les caractères du type guillemet (") ou deux points (:) ne sont pas obligatoirement demandés. L'interfaçage avec les autres modules est réalisé au moyen des variables suivantes :
- Disque qui contient le nom légal (device ou nom propre) du disque, sa présence étant assurée ;
- Nom qui contient le nom propre sous sa forme littérale et non légale ;
- Used qui contient la taille utilisée en octets ;
- Free qui contient la taille disponible.

APPEL DU PROGRAMME

Imaginons que ce programme ait été placé en RAM: avec le nom Repertoire. Le disque en DF1: existe et se nomme Disque Essais (en 2 mots pour la cause).
Les commandes suivantes sont directement acceptées :

```
RX ram:repertoire DF1:
RX ram:repertoire DF1
Rx ram:repertoire Disque essais
```

Les commandes suivantes provoquent une proposition de ce qui existe :

```
RX ram:repertoire
RX ram:repertoire d (ou tout autre nom inexistant)
Rx ram:repertoire Disqueessais (ici en un seul mot)
```

Enfin, certaines formes folkloriques sont corrigées : "DF1 :" ou encore "::". Dans ce domaine, il est difficile d'être exhaustif tant l'ingéniosité, éventuellement maladroite, est variée. Quant à l'ingéniosité maligne elle est carrément infinie ! Il faut donc se limiter aux cas les plus probables.

SAISIE DU PROGRAMME "REPERTOIRE"

Le listing source du programme est proposé en fin d'article ; les lignes ont été numérotées pour en faciliter l'analyse. Lors de la saisie du programme, il ne faudra bien entendu pas saisir les numéros de lignes.

ANALYSE DU PROGRAMME

Certaines instructions sont suffisamment documentées dans le listing. D'autres nécessitent un complément.

Ligne 3 :

- si l'invocation contient un paramètre, il est placé dans la variable Disque.

Ligne 4 :

- comme la variable Disque va être traitée, on mémorise ce qui a été réellement tapé par l'opérateur.

Ligne 5 :

- ARexx permet d'émettre une commande vers l'hôte, ici le DOS. Il suffit d'informer ARexx par ADDRESS COMMAND et d'encadrer la commande par des apostrophes ('). Ici, on crée en Ram un fichier nommé liste qui contiendra le résultat de la commande DOS Info. C'est un fichier texte.

Ligne 6 :

- ARexx ne touche pas directement un fichier. On crée un fichier tampon nommé infos qui permettra les transactions avec le fichier réel précedemment créé en Ram. Le paramètre 'R' indique qu'on effectuera seulement une lecture. Pour un écriture, on indiquerait 'W'.

Ligne 7 :

- le fichier en Ram est un fichier texte. On va donc pouvoir le lire ligne à ligne avec READLN(). Chaque ligne sera placée dans un tableau nommé Ligne. La forme WHILE ~EOF('infos') veut dire : "tant que la fin du fichier tampon 'infos' n'est pas atteinte". Enfin, BY 1 indique le pas d'avance de la fonction DO.
En ligne 9, on trouve la fin classique de la boucle. La forme DO est particulièrement riche en paramètres.

Ligne 8 :

- on lit chaque ligne du fichier infos, et on la place dans le tableau ligne.x, l'incrément étant réalisé par la boucle.

Ligne 10 :

- ce n'est pas nécessaire, mais c'est un bon usage que de placer en ligne.0 le nombre de lignes du tableau.

Lignes 11 et 12 :

- c'est aussi un bon usage que de laisser une place nette en fermant les fichiers dès qu'ils deviennent inutiles.

Ligne 14 :

- l'instruction PRAGMA est assez ésotérique. Elle permet en général de définir le répertoire courant, mais sert ici à "désactiver" le Workbench de sorte qu'il ne présente pas de requester lorsqu'on demandera un disque qui n'existe pas. Le retour au Workbench s'effectuera ligne 47.

Lignes 15, 16 :

- ouverture des boucles de traitement du nom de disque fourni. Leurs fins respectives sont en lignes 38 et 19.

Ligne 18 :

- on peut appeler un sous-programme de deux façons. La première, celle utilisée ici, suppose qu'on n'attend pas de réponse du sous-programme. L'autre forme est utilisée en ligne 36.

Ligne 20 :

- l'instruction STRIP ôte d'une chaîne donnée, soit les différents caractères définis en fin de parenthèses, soit les "blancs" du début, de fin, ou des deux comme ici. C'est utile pour les comparaisons car les blancs, bien réels, ne sont pas visibles. Ici, on essaye de parer aux excentricités de frappe en extrayant le texte du nom du disque seulement. Pour en faire un nom légal, il faut concaténer le caractère ':' en fin de chaîne exactement. C'est le rôle de l'opérateur de concaténation ||. On trouve ici un exemple de la possibilité d'associer plusieurs instructions à l'intérieur d'autres.
Enfin, il existe une autre méthode pour "nettoyer" une chaîne, par l'intermédiaire de l'instruction COMPRESS. La différence entre STRIP et COMPRESS réside dans le fait qu'alors que STRIP peut agir en début et fin de chaîne de caractères, COMPRESS peut agir aussi à l'intérieur.

Ligne 21 :

- teste l'existence du disque demandé. S'il existe, application de la branche THEN DO, sinon saut à ELSE en ligne 32.

Ligne 22 :

- analyse de la ligne i avec tri par variable, par position absolue, et élimination des variables de la ligne non souhaitées par l'usage du point ".".

Ligne 23 :

- calcul de la taille en octets à partir de la variable Used lorsqu'elle est réputée numérique ('NUM').

Ligne 24 :

- la forme, de la ligne 24 à la ligne 28, est obligatoire. Mais derrière le WHEN... THEN on peut placer un DO... END par exemple qui font de SELECT une instruction de sélection puissante. OTHERWISE permet d'en sortir gracefully si aucune des conditions n'est vérifiée. On l'utilise ici pour déterminer si le nom du disque est un device ou un nom propre.

Ligne 30 :

- pour sortir des boucles avant leur fin officielle, il y a deux méthodes : LEAVE(variable_optionnelle), qui permet de sortir d'une boucle itérative nommée, et BREAK, qui sort de la boucle la plus intérieure d'une imbrication de boucles.

Ligne 32 :

- alternative négative du test IF de la ligne 21. A ce propos, dans les boucles imbriquées de IF... ELSE, il est prudent de toujours placer une alternative ELSE même si elle n'a pas d'utilité fonctionnelle dans le programme. Dans ce cas, on lui associe NOP (comme dans ELSE NOP) qui précise qu'il n'y a rien à effectuer, mais qui permet à l'interpréteur de retrouver ses niveaux de IF.

Ligne 51 :

- c'est la définition d'un sous-programme. Ici, l'emploi est simple, mais on peut aussi envoyer des paramètres. On peut également définir deux types de variables utilisées dans le sous-programmes : celles qui sont communes à l'ensemble du programme, c'est le cas par défaut ; et celles qui sont propriété du sous-programme définies par le terme PROCEDURE.
La différence consiste dans le fait que les variables ainsi définies ne sont accessibles que par le sous-programme lui-même. Nous y reviendrons plus en détails, lorsque nous étudierons la suite de ce programme.

```
1   /**/                                        /*c'est un programme ARexx*/
2   /*TRACE  INTERMEDIATES*/                  /*pour faire un tracé oter /* */*/
3   PARSE UPPER ARG disque                  /*attend un nom de disque àl'appel*/
4   dsk=disque                               /*mémorise le nom d'appel telquel*/
5   ADDRESS COMMAND 'INFO > RAM:liste'          /*émet une commande DOS*/
6   x=OPEN('infos','RAM:liste','R')         /*ouvre le fichier RAM:liste*/
7   DO i=1 BY 1 WHILE ~EOF('infos') /*lit le fichier LISTE qui est en*/
8   ligne.i=READLN('infos')                  /*RAM jusqu'a la fin et le place*/
9   END                                      /*dans le tableau ligne indicé i*/
10  ligne.0=i-1             /*place le nombre de lignes du tableau en 0*/
11  CLOSE('infos')                                    /*ferme le fichier*/
12  ADDRESS COMMAND 'DELETE RAM:liste'       /*laisse une place propre*/
13  taquet=0                                 /*indicateur de service*/
14  PRAGMA('W','NULL')              /*interdit les réactions du Workbench*/
15  DO WHILE taquet=0               /*boucle principale , fin en ligne 38*/
16  DO WHILE disque=''/*cas où aucun nom de disque n'est donné fin 19*/
17  SAY;SAY                      /*sauts de ligne pour présentation écran*/
18  CALL Rappel()     /*appel a sous-programme sans attente de réponse*/
19  END                             /*fin de la boucle commencée en 16*/
20  disque = STRIP(STRIP(disque,'B'," ':"),'B',' "')||':'
                                                 /*ote certains caractères*/
21  IF EXISTS(disque) THEN DO i=1 TO ligne.0     /*vérifie la présence*/
22  PARSE UPPER VAR ligne.i device 5 . 15 used 23 free . 53 nom /*trie*/
23  IF DATATYPE(used)='NUM' THEN  used=used*512   /*calcule la taille*/
24  SELECT    /*    détermine si c'est un nom ou un device. fin en 28*/
25  WHEN disque=STRIP(device)       THEN aa=1       /*c'est un device*/
26  WHEN STRIP(disque,'B',':"')=nom THEN aa=2          /*c'est un nom*/
27  OTHERWISE aa=0                                /*précaution de sortie*/
28  END                                        /*fin de la fonction SELECT*/
29  IF aa~=0 THEN  taquet=1         /*teste s'il est temps de terminer*/
30  IF taquet=1 THEN LEAVE                            /*si oui on sort*/
31  END                          /*fin de la boucle DO commencée en  21*/
32  ELSE DO                /*le disque demandé n'existe pas en ligne 21*/
33  SAY;SAY;SAY                        /*3 sauts de ligne cosmétiques*/
34  SAY 'Desole je ne trouve pas ce disque!'                 /*dialogue*/
35  SAY 'Voici ce que vous avez demande:  'dsk                /*rappel*/
36  x=Rappel()       /*appel à sous-programme avec attente de réponse*/
37  END                           /*fin de la boucle DO commencée en 32*/
38  END                           /*fin de la boucle DO commencée en 15*/
39  SAY''                                    /*saut de ligne cosmétique*/
40  SELECT        /*choisit la réponse selon la forme donnée au début*/
41  WHEN aa=1 THEN SAY 'OK,je lance le disque placé en' disque
                              'nommé' nom    /*la forme était le DEVICE*/
42  WHEN aa=2 THEN SAY 'OK,je lance le disque nomme' disque 'Placé
                                 en' device   /*la forme était le Nom*/
43  OTHERWISE NOP                             /*précaution de sortie*/
44  END                        /*fin de la boucle SELECT commencée en 40*/
45  SAY''                         /*saut de ligne de présentation écran*/
46  SAY 'La Taille de ce disque est: 'STRIP(free)*512+used 'octets
       dont 'used 'sont utilises'   /*indications sur le disque lu*/
47  PRAGMA('W','Workbench')             /*on remet les choses en place*/
48  EXIT                                          /*fin du programme*/
50
51  Rappel:                                    /*c'est un sous-programme*/
52  SAY 'Voici ce qui est en ligne:'                          /*dialogue*/
53  DO i=1 to ligne.0                      /*affichage du tableau LIGNE*/
54  IF i=3 | i=4 THEN SAY ' '          /*saut de ligne de présentation*/
55  SAY ligne.i                      /*écriture de la ligne courante : i*/
56  END                             /*fin de la boucle DO commencée en 53*/
57  SAY;SAY;                           /*saut de ligne de présentation*/
58  SAY'Donnez-moi un titre valide:UNIT ou NAME (SVP) ou, pour
                                              sortir: EXIT' /*dialogue*/
59  PARSE UPPER PULL disque     /*attente de la réponse de l'opérateur*/
60  IF disque='EXIT' THEN EXIT                    /*sortie sur demande*/
61  dsk=disque                          /*mémorise la forme demandée*/
62  RETURN(disque)         /*fin du sous programme avec info retournée*/
63
```

PAR F. GUEUGNON

TOOLBOX : DRACO

Les vacances étant désormais terminées, je vous propose de reprendre l'étude des langages du domaine public disponibles sur Amiga, par le langage Draco. Ce langage, pour le moins méconnu, est l'oeuvre d'un Canadien dénommé Chris Gray. Votre première question est sans doute "qu'est-ce donc que ce nouveau langage ?

Draco a été créé par Chris Gray comme un langage de programmation système (le C fut créé à l'origine pour cette même tâche), c'est-à-dire pour l'écriture de systèmes d'exploitation, de compilateurs, d'éditeurs.... Cela ne signifie pas pour autant qu'il ne peut servir qu'à cela, mais disons qu'il a été pensé dans cette optique et possède par la même une syntaxe facilitant ce genre de travaux (opérateurs binaires, manipulation de pointeurs, gestion de structures complexes, etc).
Draco possède toutes les fonctionnalités du C, exception faite de la manipulation des champs de bits et du pré-processing de macros, mais il se rapproche plus du Pascal de par la rigidité de ses structures et l'utilisation d'un typage de données fort (pas question d'assigner impunément un entier à un pointeur). Il permet ainsi d'éviter les erreurs typiques dûes à la trop grande permissivité du C.
Du point de vue syntaxique, Draco se rapproche plus du Pascal que du C. Par exemple, il utilise := pour l'affectation et = pour la comparaison. En fait, Draco a été créé pour éliminer les principales carences de ces deux langages. Il s'agit donc, selon Chris Gray lui-même, d'un langage reprenant les avantages du C et du Pascal, en éliminant leurs défauts respectifs.
Dans le petit exemple qui suit, on peut noter l'utilisation de délimiteurs de blocs en adéquation avec les syntaxes s'y rapportant (contrairement aux accolades du C et au couple BEGIN-END du Pascal). Ainsi, une procédure est délimitée par les mots-clés proc et corp ; une boucle débute par do et se termine par od... C'est un excellent moyen mnémotechnique que d'inverser la chaîne de caractères du début de boucle pour en marquer la fin.

```
proc main()void:
int i, j;

for i from 0 upto 10 do
    for j from 0 upto i do
        write(j : 2, ' ');
    od;
    writeln();
od;
writeln("C'est fini!");
corp;
```

Autre point important de cet outil de développement, la vitesse de traitement du compilateur : près de 2000 lignes par minute. Vitesse impressionnante quand comprée au temps de traitement du même programme en C par le Lattice ou l'Aztec (qui ne sont pourtant pas à proprement parler des escargots).
Du point de vue performances pures, Draco n'a rien à envier au C disponible sur Amiga. La rapidité est globalement équivalente, avec même un léger avantage pour Draco, du moins sur les courts essais que j'ai pu faire (en particulier dans le traitement des entiers, où il excelle).
Deux aspects spécifiques du Draco sont suffisamment insolites pour être présentés. D'abord, l'instruction code qui vous permet d'introduire en plein milieu du programme une instruction assembleur. Par exemple, la ligne

```
code(0x4E750000);
```

introduit une instruction assembleur RTS dans le programme.
ç
Deuxièmement, l'opérateur arobace @ qui permet d'imposer une adresse absolue à un pointeur. Bien que dangereuse (surtout dans un environnement multitâche), cette affectation permet de simuler les équivalences du Fortran.

Mais tout n'est pas rose et Draco pose un gros problème, celui du support et de la standardisation. En effet, le langage n'est actuellement supporté que par l'auteur et n'existe que sur CP/M-80 et AmigaDos. N'espérez donc pas porter votre logiciel favori écrit en Draco sur d'autres machines, à moins que vous n'ayiez envie de faire le portage du compilateur vous-même...
Toutefois, compte tenu de la modique somme d'argent nécessaire pour vous procurer le système de développement Draco, pourquoi Diantre vous en priver ?

DRACO SUR AMIGA

La dernière version disponible de Draco est la 1.2, qui se trouve sur la disquette Fish 201 (hé oui, elle date un peu). Toutefois, pour pouvoir l'utiliser pleinement, il vous faut impérativement la Fish 77 contenant la documentation complète et des exemples de programmation en Draco (NDLR : pourquoi la 77 ? Simplement parce que la première version de Draco se trouvait sur la Fish 76 !).

INSTALLATION DE DRACO

Nous allons maintenant installer Draco sur une disquette autonome. Les ingrédients indispensables pour réaliser cette opération sont tout simplement la Fish 201 et une disquette Workbench. Suivez pas à pas les étapes suivantes, et tout devrait bien se passer.

1) Effectuez une copie de votre disquette Workbench et renommez-la "Draco".
2) Pour gagner de la place, supprimez les tiroirs Utilities, Empty, Expansion et Prefs grâce à la commande Discard du menu Workbench ;
3) Ouvrez un Shell et tapez les commandes suivantes :

```
delete Draco:Fonts ALL QUIET
```

4) Si vous ne possédez qu'un seul lecteur de disquettes, il est préférable de copier en Ram: quelques commandes du répertoire c:. Pour ce faire, tapez, toujours sous Shell :

```
copy c:copy ram:
copy c:makedir ram:
copy c:cd ram:
copy c:info ram:
copy c:dir ram:
copy c:delete ram:
path ram: add
```

5) Insérez la disquette Fish 201 dans un lecteur, et tapez :

```
copy amigalibdisk201:draco/c draco:c/ ALL
makedir Draco:Drinc
copy amigalibdisk201:draco/drinc draco:Drinc ALL
makedir Draco:Drlib
copy amigalibdisk201:draco/drlib draco:Drlib ALL
makedir Draco:script
copy amigalibdisk201:draco/s draco:script ALL
```

6) Editez la startup-sequence de la disquette Draco: et remplacez la ligne

```
path ram: c: sys:utilities sys:sytem s: sys:prefs add
```

par

```
path ram: c: sys:system s: add
```

pour tenir compte de la suppression des répertoires "Utilities" et "Prefs".
7) Editez ensuite le fichier StartupII et ajoutez-y, juste avant la dernière ligne, les lignes suivantes :

```
assign Drlib: Draco:drlib   ; bibliothèques
assign Drinc: Draco:drinc   ; includes
assign Drs: Draco:script ; procédures standards de link
path Drs: add
stack 16000
```

Note : la commande stack 16000 permet d'ajouter un peu de pile système pour le compilateur. Ceci vous mettra à l'abri de certains "plantages" dûs, en général, à un débordement de la pile système lors de la compilation. 16000 est le chiffre minimum. Pour de gros programmes, il se peut que vous soyez obligé d'augmenter cette valeur.

8) Enfin, créez un répertoire 'source' où vous pourrez stocker vos sources Draco :

```
makedir Draco:source
```

Si tout s'est bien passé, vous n'avez plus qu'à réinitialiser l'Amiga sur la disquette Draco et vous pouvez sans problème commencer à utiliser l'environnement de développement Draco.

DRACO PAR L'EXEMPLE

Le meilleur moyen d'étudier le fonctionnement de Draco est de prendre un exemple des plus classiques et de voir les différentes étapes qui nous permettront de créer un programme exécutable. L'exemple choisi est bien entendu d'imprimer "bonjour tout le monde" à l'écran.
Pour écrire le source, nous allons utiliser l'éditeur Ded, fourni avec le système Draco. Vous pouvez bien entendu utiliser l'éditeur de votre choix, mais celui-ci étant fourni sur la disquette, et de plus écrit en Draco lui-même, pourquoi aller chercher ailleurs ?

Positionnez-vous dans le répertoire de travail et tapez :

```
ded bonjour.d
```

Ensuite entrez le source suivant :

```
proc main()void:
    writeln("Bonjour tout le monde");
corp;
```

Sortez de l'éditeur en sauvegardant par la séquence de touches <Esc-e>.

Compilez ce source avec la commande :

```
draco fichier1[.d] fichier2[.d]... fichierN[.d]
```

L'extension par défaut pour un fichier source Draco est .d. Chaque fichier est bien entendu compilé indépendamment des autres. Si la compilation s'est bien passée un fichier portant l'extension .r est généré pour chaque source compilé (je vous avouerai que cette extension est assez douteuse, sans doute un héritage de CP/M). Au sujet des extensions, il vous savoir que les "includes" Draco portent l'extension .g, ce qui permet de les distinguer à coup sûr des autres.

Dans notre exemple il suffit de taper :

```
Draco bonjour
```

l'extension .d étant prise par défaut.

Un fichier bonjour.r a logiquement été généré. Dans le cas où vous auriez fait une erreur en tapant l'exemple, modifiez le source et recompilez-le jusqu'à ce que le fichier objet soit généré.
On utilisera BLink pour générer un exécutable à partir du fichier objet. De nombreux exemples de procédures de link sont fournis sur la disquette Fish 201. Pour linker notre programme "bonjour", le script baptisé Draco:s/link.s, convient très bien :

```
execute link.s bonjour
```

Et là, ô miracle, un programme exécutable du nom de "bonjour" est créé. Pas mal, non ?

LE RESTE...

Deux mots sur la documentation, qui est abondante et très pédagogique (malheureusement en anglais, mais néanmoins facilement comprehensible). Je vous rappelle que la disquette 77 est nécessaire pour posséder l'ensemble de la documentation.
Du point de vue compatibilité, il n'y a pas l'ombre d'un problème puisque Draco fonctionne aussi bien sous système 1.3 que 2.0 (les tests éffectués sur Amiga 3000 se sont révélé très concluants).
Enfin, je vous signale que Draco est un logiciel Shareware, ce qui signifie que si vous l'utilisez régulièrement, vous avez le devoir moral d'envoyer la contribution demandée par l'auteur. Vous serez de plus déclaré utilisateur enregistré, ce qui vous permettra sûrement de recevoir les nouvelles versions du produit. L'adresse de l'auteur se trouve sur la disquette.

CONCLUSION

Nous sommes donc en présence d'un produit très professionnel que je ne saurais trop vous conseiller, ne serait-ce que par simple curiosité. En effet, pour le prix de revient d'un tel produit, il serait bête de s'en priver. Et puis qui sait, peut-être serez-vous, vous aussi, atteint par le virus de la programmation en Draco, virus bien inoffensif celui-là ?

Herr Doktor Von GlutenStimmelImDorf
a l'esprit Draconien.

NOTE AUX ABONNES AVEC DISQUETTE

L'environnement de développement de Draco occupant une disquette entière, voire deux avec les documentations, il nous est malheureusement impossible de placer ce langage sur la disquette d'accompagnement de ce mois-ci.

D'autant plus que de par la volonté de l'auteur, Chris Gray, Draco doit être distribué dans son intégralité... ou pas du tout. Même après être passé à la moulinette de notre archiveur préféré - LHArc pour ne pas le nommer, la taille du fichier obtenu reste encore décourageante : plus de 450 Ko, soit beaucoup plus qu'il n'en faut pour cohabiter en toute quiétude avec les autres rubriques de l'ANT. La décision fut donc prise la mort dans l'âme : Draco sera absent de notre disquette ANT26.

Rassurez-vous toutefois : comme vous avez sans doute d'ores et déjà pu le constater, la place ainsi "gagnée", si j'ose dire, est bien entendu utilisée à d'autres fins. Vous trouverez donc plusieurs utilitaires du domaine public dans un répertoire spécial, baptisé, je vous le donne en mille, DomPub.

Pour vous procurer Draco, il va donc vous falloir jouer du timbre poste et commander les disquettes Fred Fish 77 et Fred Fish 201 à une association de Domaine Public. Excellente occasion pour nous pour les recenser et créer une nouvelle "rubrique" au sein de l'ANT.

Nous avons donc établi ici une liste, non exhaustive et de loin, des associations que vous pouvez contacter. Cette liste est présentée par ordre alphabétique et donne, en plus de l'adresse et si possible du numéro de téléphone, le prix des disquettes à la pièce pour chaque association. La plupart d'entre elles vous propose un catalogue ; le prix en est éventuellement indiqué.

La liste présente également quelques serveurs Transac (Télétel 2 et 3) ou RTC (Réseau Téléphonique Commuté) sur lesquels vous pourrez trouver du Domaine Public à télécharger. Nous avons essayé, tant que possible, d'indiquer le prix du câble et du logiciel de téléchargement nécessaire (seul le logiciel diffère d'un serveur à un autre ; le câble reste identique pour tous).

A tous les responsables d'association(s) et/ou de serveur(s) qui se plaindraient de ne pas figurer dans cette liste, nous tenons à préciser qu'elle a été conçue en feuilletant les magazines Amiga Revue et AmigaNews. Nous attendons évidemment vos courriers à la rédaction pour vous y inscrire et ainsi, faire grandir cette liste au fil des mois.

La rédaction

ASSOCIATIONS

Attila PDS
BP 192
63805 Cournon Cedex
Disquette : 15 F
Catalogue : 10 F ou enveloppe auto-adressée et disquette vierge

D2P (Diffusion de Domaine Public)
1 Ter, rue Henri Barbusse
78390 Bois d'Arcy
Disquette : 15 F
Envoi recommandé : 15 F

Free Distribution Software (FDS)
Boîte Postale 134
59453 Lys Lez Lannoy Cedex
Disquette : 15 à 20 F
Catalogue : 3 timbres à 2,50 F
Envoi recommandé : 12 F
Envoi colissimo : 20 F

Load'n'Enjoy
BP 10
08000 Villers-Semeuse
Disquette : 10 F
Catalogue : 3,80 F

PDS FreeLine
7, rue de Coursic
64100 Bayonne
Tel : (16) 59.59.19.37
Disquette: 15 F
Envoi recommandé : 15 F

Phoenix DP
90, rue Dragon
13006 Marseille
Disquette : 20 F
Catalogue : 10 F (remboursé au 1er achat)
Envoi collisimo : 20 F

The Commodexplorer/Corsaire Productions
A6 La Rocade
91160 Longjumeau
Disquette : 15 F à 20 F
Catalogue : 1 timbre à 3,80 F

SERVEURS TELETEL

3615 ANT
Câble : 110 F
Logiciel : ChanelSoft - 59 F
Kit complet : 159 F

3615 COMREV
Câble : 110 F
Logiciel : ChanelSoft - 59 F
Kit complet : 159 F

3615 LOAD
Câble : 95 F
Logiciel : 45 F
Kit complet : 95 F (disquette gratuite)

SERVEURS RTC

Numéro 6
Tel : 43.31.79.53
Logiciel : FlamTrans

Khéops
Tel : 45.07.85.03
Logiciel : FlamTrans

SGT Flam
Tel : 39.55.84.59
Logiciel : FlamTrans

Valinor
Tel : 45.07.85.03
Logiciel : FlamTrans

DEMOS : LE TRIAL PLAYFIELD!

Voici un titre pour le moins alléchant, qui a été imaginé dans le but d'attirer le lecteur et le forcer à lire cet article, même contre sa propre volonté.

Un article que j'ai ecrit à la sueur de mon front, sous le soleil qui commence à taper si fort qu'il dérègle ces trois neurones deprimés d'avoir échoué dans mon crâne bouillonnant... Bon, sérieusement, je vais tenter de vous expliquer comment obtenir un pseudo trial-playfield, c'est-à-dire trois plans totalement indépendants les uns des autres. Vous connaissez déjà les possibilités de deux de ces trois plans, qui sont ceux décrits dans toutes les docs, au chapître Dual Playfield. Mais je vais quand même vous en faire un petit résumé.

1 + 1 = 2

La gestion de chaque playfield est séparée en ce qui concerne la taille et le scroll, mais pas la couleur. En effet, l'Amiga n'a, à ma connaissance, qu'un seul registre permettant d'indiquer à la machine le nombre de bitplans de l'écran (il s'agit bien sûr de BPLCON0). Or, ayant deux playfields et un seul compteur de bitplans, la machine tranche le problème de la manière suivante : si le nombre de bitplans est pair, les deux playfields auront la même quantité de plans. Dans le cas contraire, c'est le playfield 1 qui bénéficiera du bitplan suplémentaire.

ET DE TROIS !

Pour en revenir au fameux troisième playfield, celui-ci n'étant pas "officiel", il est bien moins souple et ne permet que le monochrome. De plus, les scrollings verticaux et horizontaux sont plus compliqués à programmer que les scrollings normaux.

Il est entièrement composé de sprites.
- "Mais.. mais.. comment fait-il ? Les docs que nous avons lues affirment que l'on ne peut pas afficher plus de 8 sprites sur une même ligne de raster !" crie la foule d'une seule voix. Et moi de surenchérir :
- "J'irais même plus loin, en affirmant qu'il n'est composé que d'un seul sprite", tout fier d'avoir enfin quelqu'un qui m'écoute, qui me fait momentanément oublier que je ne suis qu'une poussière au milieu de ce grand tout que représente l'Univers. La technique est très simple, en fait ; elle fait beaucoup intervenir le Copper et les sprites.

VOYAGE AU PAYS DES SPRITES

Comme vous le savez sûrement, le procédé habituel pour se servir des sprites est de faire pointer les canaux DMA adéquats (SPRxPT) sur une partie de la Chip-Ram où l'on aura préalablement placé les données du sprite. Rappel : ces données indiquent sa position (SPRxPOS et SPRxCTL) ainsi que l'image du sprite elle-même (SPRxDATA et SPRxDATB). Pour que le sprite apparaisse à l'écran, il faut aussi autoriser le DMA sprite en mettant à 1 le bit concerné (bit SPREN dans DMACON). Dans le listing qui suit cet article, une chose saute aux yeux et les mord violemment : le DMA sprite n'est pas autorisé ! En effet, et c'est là tout le secret, j'utilise le mode dit manuel des sprites, c'est-à-dire que je charge le Copper de faire le travail du DMA sprite, travail qui consiste à mettre la position et l'image (ce terme me paraît légèrement exagéré d'un seul coup car en fait, il s'agit un seul et unique mot de 16 petits bits) la position et l'image du sprite, disais-je donc, dans les registres prévus à cet effet. Pour obtenir un playfield continu, il faut renouveler le sprite tout les 16 pixels. J'effectue cette operation ô combien délicate avec l'aide précieuse du Copper. Comme chacun sait, chaque instruction Copper prend un temps machine équivalent à la durée d'affichage de 8 pixels en basse résolution ; donc, la douloureuse vérité s'impose : on doit renouveler le sprite en ne changeant que deux registres au maximum. Ce fut pénible, mais celà devait être écrit.

Le problème vient du fait que d'un côté, les sprites sont caractérisés par quatre registres (SPRxPOS, SPRxCTL, SPRxDATA et SPRxDATB) et que de l'autre côté, on ne peut en changer que deux. Pour contourner celà, on peut facilement arrêter d'utiliser SPRxDATB et donc se contraindre à avoir un sprite monochrome. On remarque aussi que SPRxCTL reste le même durant toute la longueur de la ligne de raster. Donc, si on ne le modifie qu'en début de ligne, il ne nous reste plus, ô miracle, que deux registres à bidouiller, à savoir SPRxPOS et SPRxDATA.

CONCRETEMENT

La routine build_coplist, qui construit - comme son nom l'indique - une CopperList capable de réaliser toute les merveilles dont je viens de vous parler, procède ainsi : à chaque début de ligne, on initialise SPRxCTL (qui sera constant pendant toute la ligne) avec le numéro de la ligne actuelle plus 1, puis on attend avec un Wait aproprié que le rayon atteigne le bord gauche de l'écran et on commence la succession de SPRxPOS et SPRxDATA (20 de chaque en tout). Et ainsi de suite pour toutes les lignes.

Maintenant que vous connaissez le principe du pseudo troisième playfield, je vais vous expliquer comment on peut faire des scrolls dessus. Déjà, les techniques simples et habituelles tel que faire bouger un pointeur sur un bitplan plus grand que l'écran ne sont pas utilisables, car toutes les positions des sprites (registre : SPRxPOS) sont absolues, donc changer un pointeur ne donnerait rien de très intéressant. Résultat, l'unique méthode de scroll que je connaisse s'adaptant à ce cas se résume à un grand coup de Blitter, ce qui permet de faire un scroll vertical très facilement. Pour ce qui est du scroll horizontal, c'est plus hardu.

Dans le cas particulier (et rare) où le pas du scroll est de 16 pixels, on peut encore utiliser un déplacement de mémoire simple à l'aide du Blitter. Pour le cas d'un pas inférieur à 16 pixels, il faut encore utiliser le Blitter mais cette fois-ci beaucoup plus finement, en se servant du fait que ce qui sort d'une ligne, à cause du glissement, reparaît sur le début de la ligne suivante.

Pour être honnête, je ne suis pas du tout l'inventeur de cette technique, que je connais grâce à une cartouche que je ne nommerais pas et qui me permet de décoder une CopperList à n'importe quel moment d'un jeu ou autre, de la modifier et de reprendre le cours des choses... Elle a été utilisée dans des jeux tels que Turican II ou SwitchBlade 2 pour afficher les scores en transparent. Le nom de l'auteur de la première routine m'est complètement inconnu, mais je le remercie ici publiquement.

Le mois prochain, vous aurez dans cette rubrique probablement la technique de scrolling la plus rapide possible (et en plus c'est vrai !). Il est en fait virtuellement impossible de faire mieux. Encore un algo que je n'ai pas trouvé, d'ailleurs. Son auteur est à ma connaissance Dino Dini, le génie qui a pondu Kick Off.

Au fait, n'attendez plus la fin du monde ! Je l'ai vu, un vulgaire gnab gib.

```
** LISTING DE JEROME ETIENNE POUR L'ANT
** sur le 'trial playfield'

    incdir 'include:'
    include    'exec/exec_lib.i'
    include    'hardware/custom.i'

custom =   $dff000
execbase   =   4

plane_x    =   320
plane_y    =   200
plane_width_word =   plane_x/16
plane_size=   (plane_x/8)*plane_y

vsync: macro
.wait_vsync@:
    move.l vposr(a5),d0
    and.l  #$1ff00,d0
    cmp.l  #$03000,d0
    bne.s  .wait_vsync@
    endm

wait_blt: macro
    btst   #14,dmaconr(a5)
.loop_wait_blt@:
    btst   #14,dmaconr(a5)
    bne.s  .loop_wait_blt@
    endm

*****************************************************
**************  programme principal  ****************
*****************************************************
    move.l (execbase).w,a6
    lea custom,a5

    CALLEXEC   Forbid
    move.w #$03e0,dmacon(a5)* all dma off except disk

    move.w #$1200,bplcon0(a5)    * 1 SEUL bitplan
    clr.w  bplcon1(a5)
    clr.w  bplcon2(a5)
    clr.w  bpl1mod(a5)
    clr.w  bpl2mod(a5)
    move.w #$2981,diwstrt(a5)    * 320*200
    move.w #$f1c1,diwstop(a5)
    move.w #$0038,ddfstrt(a5)
    move.w #$00d0,ddfstop(a5)


    bsr build_coplist

    move.l coplist_adr,cop1lc(a5)   * > run my coplist
    clr.w  copjmp1(a5)    */

    move.w #$83c0,dmacon(a5)* dma blitter,copper & bitplane on

main_loop:

    vsync

    wait_blt      *
    clr.w  bltadat(a5)    * >-efface l'ecran a chaque vbl
    clr.w  bltamod(a5)    * > pour bien montrer que le
    move.w #$ffff,bltafwm(a5)    * > playfield est dans le sprite
    move.w #$ffff,bltalwm(a5)    * > et non dans le bitplan
    move.w #$01f0,bltcon0(a5)    * >
    clr.w  bltcon1(a5)    * >
    clr.w  bltdmod(a5)    * >
    move.l plane_adr,bltdpt(a5) */
    move.w #plane_y*64+plane_width_word,bltsize(a5)

    move.b $bfec01,d0 *
    not d0  * >capture the key wich is pressed
    ror.b  #1,d0  */ and put is code RAW in d0
    cmp.b  #$45,d0    *
    beq.s  init_end   */ sort si on press sur esc

    btst   #6,$bfe001
    bne.s  main_loop

******** init end ***********
*   reactivation de l'old coplist
init_end:
    wait_blt
    lea GfxName(pc),a1    * nom de la library ds a1
    moveq  #0,d0  * version 0 (the last)
    CALLEXEC   OpenLibrary    * lib graphique ouverte
    move.l d0,a4   * adr de graphicbase ds a4
    move.l 38(a4),cop1lc(a5)* chargement de l'adr de
    clr.w  copjmp1(a5)    * l'old coplist et lancement

    move.w #$83e0,dmacon(a5)*    activation    des    canaux    dma
necessaires
    CALLEXEC   Permit * multi switching autorise

fin:   moveq  #0,d0  * flag d'erreur desactive
    rts
GfxName:   dc.b    "graphics.library",0
    even
*****************************************************
build_coplist:
    move.l coplist_adr(pc),a0
    move.w #bplpt,(a0)+  *
    move.w plane_adr(pc),(a0)+  * >init les pointeurs bitplans
    move.w #bplpt+2,(a0)+    * >
    move.w plane_adr+2(pc),(a0)+    */
    lea image,a1
    moveq  #$29,d0
.loop_chaque_ligne:
    move.w #spr+sd_ctl,(a0)+*
    move.w d0,d1   * >-init spr_ctl a la ligne juste
    lsl.w  #8,d1   * > superieure a la ligne actuelle
    and.w  #$ff00,d1  * >
    move.w d1,(a0)    * >
    addq.w #1,(a0)+   */
    move.w d0,d1   *
    ror.l  #8,d1   * >-init le wait juste sur le bord
    and.l  #$ff000000,d1 * > gauche de l'ecran
    or.l   #$0039fffe,d1 * >
    move.l d1,(a0)+   */
    move.w d0,d1   *
    lsl.w  #8,d1   * >-prepare le futur spr_pos
    and.w  #$ff00,d1  * >
    or.w   #$0040,d1 */
    moveq  #20-1,d2   * 20= largeur de l'ecran /16
.loop_each_word:
    move.w #spr+sd_pos,(a0)+* init spr_pos
    move.w d1,(a0)+   */
    move.w #spr+sd_dataa,(a0)+  * init spr_data
    move.w (a1)+,(a0)+    */
    addq.w #8,d1   *on calcul le prochain spr_pos et
    dbf d2,.loop_each_word    */ boucle pour le prochain mot
    addq.w #1,d0   * -on teste si on est arrive sur la
    cmp.w  #$f1,d0    * > ligne 200.Si oui, on a fini.
    blt.s  .loop_chaque_ligne    */

    move.l #$fffffffe,(a0)+ * montre la fin de la clist
    rts

******** datas
*   variables
plane_adr:dc.l    ecran
coplist_adr: dc.lcoplist

; image 320x200, 1 bitplane (format RAW)
image_raw:incbin 'fnt_32.bit'
image  =   image_raw+64
    section    ZONE_CHIP,BSS_C
ecran: ds.l    plane_size/4
coplist:   ds.l    34000/4

    end
```

Par J.ETIENNE

L'INUTILITAIRE DU MOIS : New

Histoire de changer un peu, nous allons vous présenter un programme qui présente la particularité d'être parfaitement inutile. Quoique, en y regardant de plus près...

Judicieusement baptisé NewPic, ce programme se propose de remplacer la morne et triste main de l'Amiga - vous savez, celle qui demande une disquette Workbench - par une image IFF de votre choix. Si le résultat est purement esthétique, la mise en oeuvre demande quelques connaissances des processus de Reset et de Boot de l'ordinateur, un tel programme devant être capable de résister au Reset pour resurgir inlassablement à chaque redémarrage.

SEQUENCE FRISSON

Or donc, lorsque vous appuyez presque machinalement sur le bouton on/off du boîtier d'alimentation, tout un tas de choses auxquelles on n'auraient même pas pensé se passent. C'est ce que l'on appelle le démarrage à froid : d'abord, l'électronique chauffe un peu, histoire de pouvoir travailler dans de parfaites conditions. Ensuite, le logiciel prend la main et commence un véritable travail de titan, qui consiste à s'informer de la configuration matérielle et à initialiser toutes les couches logicielles du système d'exploitation.

D'un autre côté, une autre séquence de démarrage, semblable à la première pour l'essentiel mais pas identique, survient lors des démarrages à chaud, c'est-à-dire essentiellement lors d'un Reset, causé soit par l'appui simultané sur les maintenant célèbres trois touches Control-Amiga gauche-Amiga droite, soit par la volonté d'un programme quelconque, soit enfin suite à un plantage de la machine. Ce qui se passe à ce moment là est fort intéressant, et je vous propose de le découvrir avec moi.

SEQUENCE DEMARRAGE

Il faut savoir que lorsque le micro-processeur 68000 se trouve en état de Reset, un système de recouvrement mémoire fait apparaître une image de la Rom en bas de mémoire. Ceci est nécessaire parce que justement, le 68000 va chercher l'adresse de la première instruction à exécuter à l'adresse $00000004 (heu... 4 en décimal) et la valeur initiale du pointeur de pile à l'adresse $00000000. Or, si de la Ram occupe cet espace, aucune valeur valide ne pourra y être trouvée, d'où le recouvrement évoqué plus haut. Le 68000 va chercher les informations en ce qu'il croit être l'adresse 0, alors qu'il va sans même s'en rendre compte les trouver en $FC0000.

Bref, le 68000 se retrouve maintenant en $FC00D2, où après quelques tests internes (intégrité de la Rom, version du système), il supprime le recouvrement mémoire (pour les curieux, c'est le bit 0 du port A du CIA-A qui le contrôle). La Ram est maintenant à nouveau disponible en $00000000 et jusqu'à $0007FFFF (c'est la Chip-Ram ; aucune extension mémoire n'est encore reconnue) et la table des auto-vecteurs du 68000 peut être mise en place. Ensuite, on construit ExecBase, on place son adresse en $00000004, et on appelle la routine de démarrage à chaud (je simplifie un peu, mais c'est ça).

Le démarrage à chaud consiste dans un premier temps à vérifier l'intégrité de la structure ExecBase. Si ce test échoue, un démarrage "à froid" est provoqué. Un peu plus tard, les premiers programmes "résidents" (entendez par là, Reset-proof) sont appelés, via le vecteur ColdCapture. Plus tard encore, une fois que la pile aura été initialisée, c'est CoolCapture qui sera appelé. Après ça, Exec recherche en mémoire toutes les structures résidentes et leur passe le contrôle du processeur, le temps de leur initialisation. C'est là qu'il nous faudra chercher le moyen de devenir parfaitement résident, dans embêter les détecteurs de virus qui recherchent ces affreuses bébêtes dans ColdCapture et CoolCapture, comme VirusX par exemple. Malheureusement, d'autres anti-virus recherchent également les structures résidentes et les éliminent impitoyablement de la mémoire si elles ont le malheur de ne pas pointer sur une adresse en Rom. Contre ceux-là, impossible de se protéger.

SEQUENCE STRUCTURE

Comme vous devez commencer à en avoir l'habitude, un module résident est défini par une structure que l'on peut trouver dans "exec/resident.i" :

```
STRUCTURE RT,0
    UWORD  RT_MATCHWORD  ; mot de reconnaissance
    APTR   RT_MATCHTAG   ; pointeur sur ce mot
    APTR   RT_ENDSKIP; fin de la structure résidente
    UBYTE  RT_FLAGS
    UBYTE  RT_VERSION
    UBYTE RT_TYPE ; type du module
    BYTE   RP_PRI ; priorité du module
    APTR   RT_NAME    ; nom du module
    APTR   RT_IDSTRING   ; chaîne de description
    APTR   RT_INIT    ; routine d'initialisation
    LABEL  RT_SIZE
```

avec

```
RTC_MATCHWORD EQU $4AFC ; mot de reconnaissance (constante)
```

Le premier mot est donc la constante $4AFC qui permet à Exec de reconnaître une structure résidente lorsqu'il parcourt la Ram. Le second élement, RT_MATCHTAG, pointe sur la structure elle-même (donc, sur RT_MATCHWORD) et sert de deuxième vérification quant à la validité de cette structure. Parmi les champs réellement importants, on trouve également RT_ENDSKIP, qui indique la fin de la structure. Ce pointeur permet de la rallonger à volonté, afin par exemple de maintenir résidentes d'autres données. En général, le programme résident sera lui aussi logé entre la structure et l'adresse pointée par RT_ENDSKIP. L'adresse de ce programme est placée dans le champ RT_INIT.

La priorité de la structure indique dans quel ordre elle sera initialisée. Plus la priorité est élevée, plus l'initialisation arrivera tôt. Par exemple, exec.library présente une priorité de 120, celle du trackdisk.device est de 14, celle du strap (le module qui affiche la main Workbench et charge le boot-block de la disquette en DF0:), de -60 et celle de la dos.library, de 0. En général, on utilisera une priorité négative, afin d'être sûr que le système complet est initialisé. J'ai arbitrairement choisi -59, au cas où d'autre programmes résidents voudraient s'intaller avant la lecture du boot-block, mais -1 aurait amplement suffit. Attention toutefois à ne pas descendre en dessous de -59, étant donné que le strap ne rend pas la main ; c'est toujours le dernier module appelé.

Seuls deux flags sont possibles : d'abord, RTF_COLDSTART que l'initialisation de la structure devra être effectuée devinez quand ? Lors du démarrage à froid, oui. Ensuite, RTF_AUTOINIT indique que RT_INIT pointe non pas sur une routine personnelle, mais sur une table de données qu'Exec initialisera en appelant la fonction InitStruct(). Celà est plus particulièrement utile dans le cas de bibliothèques et de devices et ne nous intéresse pas aujourd'hui. Nous nous contenterons donc de positionner RTF_COLDSTART.

Une dernière remarque, mais de taille, concernant les structures résidentes : elles doivent absolument se trouver en Chip-Ram pour avoir une chance d'être reconnues. Un bug, ou plutôt une omission dans Exec limite en effet la recherche à la zone de données $00000000 à $0007FFFF.

SEQUENCE RESIDENTE

Pour une fois, Exec ne présente aucune fonction destinée à rendre résidente une structure. Il faut tout faire par soi-même. Rassurez-vous, celà n'est pas très compliqué.

On utilise pour celà les champs KickMemPtr, KickMemTag et KickCheckSum de la structure ExecBase. KickMemPtr pointe sur une liste de structures MemList (NDLR : voire l'excellent article de Philippe

Vautrin à ce sujet dans notre dernier numéro) dont on comprend vite l'utilité : dès que son gestionnaire de mémoire est prêt, c'est-à-dire bien avant la recherche et l'initialisation des structures résidentes, Exec alloue, via AllocAbs(), toutes les zones désignées par ces MemLists, celà pour éviter qu'elles ne soient allouées par d'autres, au risque d'écraser les données qui s'y trouvent. Celà n'est bien entendu utile que pour les modules résidents en Ram. Donc, pour se protéger contre toute atteinte à notre vie privée, nous devrons initialiser correctement une MemList et l'insérer dans ce tableau.

KickTagPtr est un pointeur sur une liste des structures résidentes trouvées lors du démarrage à froid. Ensuite, lors d'un démarrage à chaud, Exec recherchera les structures résidentes via cette liste, et uniquement ainsi. Donc, pour être sûr qu'au prochain Reset notre structure soit toujours présente, il faudra nous y insérer nous-même. Nous verrons un tout petit plus loin le format un peu particulier de cette liste.

KickCheckSum est devinez quoi ? Une somme de contrôle, oui, qu'Exec utilise pour tester l'intégrité des données pointées par KickMemPtr et KickTagPtr. Si ce checksum est incorrect, un démarrage à froid est provoqué, et adieu le petit programme résident. Or, en nous insérant dans ces deux listes, nous faussons le checksum. Il faudra donc le recalculer, à l'aide de la fonction SumKickData().

Enfin, un quatrième champ d'ExecBase, falcutatif celui-là et nommé ResModules, peut être utilisé. Il pointe également sur une liste des structures résidentes en mémoire, aussi bien Rom que Ram, et est utilisé par la fonction FindResident() pour rechercher un module par son nom. Si on ne s'insère pas dans cette liste, FindResident() sera incapable de trouver notre module, mais celui-ci sera tout de même bel et bien là, et actif de surcroît.

Les formats des listes pointées par KickTagPtr et ResModules sont absolument identiques : une entrée nulle indique la fin de la liste et une entrée avec le bit 31 positionné indique qu'il s'agit d'une pointeur sur une seconde liste (il faut donc effacer ce bit 31 pour obtenir l'adresse de la nouvelle liste). Toute autre valeur est considérée comme l'adresse d'une structure résidente.

Comment faire son trou dans toutes ces listes ? Deux méthodes sont possibles : soit on recherche la première entrée vide, qui représente donc la fin de la liste, et on y insère un pointeur sur une liste créée par nous-même et qui ne contient que notre structure ; soit on s'insère directement au premier rang en faisant pointer KickTagPtr sur nous-mêmes. Notre liste à nous contiendra dans ce cas deux pointeurs : le premier sur notre structure résidente, le second sur la liste que pointait originellement KickTagPtr. Le raisonnement est bien entendu strictement identique en ce qui concerne ResModules.

SEQUENCE BOOT-BLOCK

Nous en arrivons maintenant au point où notre code résident est appellé par Exec. Je vous rappelle qu'à ce stade, tout le système est correctement initialisé, et qu'on peut donc appeler la graphics.library pour afficher la nouvelle image (en créant non pas un écran au sens Intuition du terme, mais un bon vieux View et tutti quanti) et le trackdisk.device pour lire le boot-block.

Une fois le boot-block lu, on vérifie qu'il s'agit bien d'une disquette bootable et si c'est le cas, on termine gentiment notre programme, après avoir libéré toutes les resources allouées. Le véritable strap sera alors appelé à son tour ; il trouvera une disquette bootable dans le drive et n'aura pas besoin d'afficher sa petite main potelée. Le reste ne concerne plus que lui. Tiens, juste une précision pendant que j'y pense : saviez-vous que cette fameuse main n'est pas une image au sens propre du terme, gravée dans la Rom, mais un dessin réalisé par une suite de Move(), Draw(), FloodFill() et autres BltTemplate() ? Etonnant, non ?

Max
n'a pas ses mains sans ses poches

LISTING 1

```
;
; NewPic - Remplace la main du WB par une image IFF au choix.
;  1991, Max pour ANT
;
; Usage : 'NewPic I image'  pour installer
;         'NewPic R'        pour dé-installer
;

; ********************************************************
; * ATTENTION : Un minimum de vérifications sont faites *
; *             sur la validité de l'image IFF.          *
; ********************************************************

    opt o+,ow-

Ç   incdir  "include:"
    include "exec/memory.i"
    include "exec/ports.i"
    include "exec/resident.i"
    include "exec/execbase.i"
    include "devices/trackdisk.i"
    include "devices/bootblock.i"
    include "graphics/gfx.i"
    include "graphics/view.i"
    include "graphics/gfxbase.i"
    include "libraries/dos.i"

    include "exec/exec_lib.i"
    include "graphics/graphics_lib.i"
    include "libraries/dos_lib.i"

; ************************************
CALL    MACRO
    jsr _LVO1(a6)
    ENDM

DEFBSTR MACRO
.len@   dc.b    .end@-.str@
.str@   dc.b    1,10
.end@   even
    ENDM

; Codes d'erreur renvoyés par LoadPic
E_NOERR EQU 0   ; Pas d'erreur à signaler
E_FILE  EQU 1   ; Fichier non trouvé
E_IFF   EQU 2   ; Pas un fichier IFF
E_ILBM  EQU 3   ; Pas un fichier ILBM
E_BAD   EQU 4   ; Fichier IFF corrompu

; ************************************
    rsreset
args    rs.l    2       ; Variables du programme principal
DosBase rs.l    1
stdout  rs.l    1
picadr  rs.l    1
readbuf rs.b    12
VARSIZE rs.w    0

    rsreset
MyMem   rs.b    0           ; Variables du programme résident
mm_port rs.b    MP_SIZE     ; MsgPort
mm_tdio rs.b    IOTD_SIZE   ; IoExtED
mm_buff rs.b    2*TD_SECTOR ; Buffer pour le trackdisk.device
mm_chng rs.l    1           ; Nb de changements de disquette
gfxbase rs.l    1           ; GfxBase
mm_oldv rs.l    1           ; OldView
mm_view rs.b    v_SIZEOF    ; View
mm_vp   rs.b    vp_SIZEOF   ; ViewPort
mm_bmap rs.b    bm_SIZEOF   ; BitMap
mm_rinf rs.b    ri_SIZEOF   ; RasInfo
mm_SIZE rs.w    0

; ************************************
Start   movea.l $4.w,a6
    lea VARS(pc),a5
    movem.l d0/a0,args(a5)

    lea dosname(pc),a1
    moveq   #0,d0
```

```
    CALL    OpenLibrary
    move.l  d0,DosBase(a5)
    beq.s   NoDos
    movea.l d0,a6
    CALL    Output
    move.l  d0,stdout(a5)
    movem.l args(a5),d0/a0
    clr.b   -1(a0,d0.w)
    move.b  (a0)+,d0
    beq.s   NoArgs
    andi.b  #$df,d0     ; Conversion majuscule
    cmpi.b  #'I',d0     ; Installation demandée ?
    beq.s   InstallTag
    cmpi.b  #'R',d0     ; Dé-installation ?
    beq RemoveTag
NoArgs  lea usage.txt(pc),a0
    bsr Print
Exit    movea.l DosBase(a5),a1
    movea.l $4.w,a6
    CALL    CloseLibrary
NoDos   moveq   #0,d0
    rts

; ************************************
InstallTag:
    move.l  a0,-(sp)
    bsr FindMe      ; Déjà intallé ?
    beq.s   InstallIt   ; non
    addq.l  #4,sp
    lea deja_la.txt(pc),a0
    bsr Print
    bra.s   Exit

InstallIt:
    movea.l (sp)+,a0
.space  cmpi.b  #' ',(a0)+
    beq.s   .space
    subq.l  #1,a0
    bsr LoadPic     ; Charge l'image IFF
    beq.s   .PicOk
    lea erreurs(pc),a0
    subq.l  #1,d0       ; Le code d'erreur sert d'indice
    lsl.l   #2,d0       ; dans le tableau
    movea.l 0(a0,d0.l),a0
    bsr Print
    bra.s   Exit

.PicOk  movea.l $4.w,a6     ; Alloue la mémoire nécessaire
    move.l  #RESSIZE,d0
    moveq   #MEMF_CHIP|MEMF_PUBLIC,d1  ; CHIP !!!
    CALL    AllocMem
    tst.l   d0
    bne.s   .MemOk
    lea memoire.txt(pc),a0
    bsr Print
    bra.s   Exit

.MemOk  movea.l d0,a4       ; a4 pointe cette mémoire
    lea RomTag(pc),a0
    lea (RESSIZE).w,a1  ; Copie les données du programme
    exg d0,a1       ; dans la mémoire réservée
    CALL    CopyMem

    CALL    Disable     ; Qu'on ne me dérange pas !

    move.l  a4,RT_MATCHTAG(a4) ; Le RomTag pointe sur lui
    lea RESSIZE(a4),a0
    move.l  a0,RT_ENDSKIP(a4)  ; Fin des données résidentes
    lea TagName-RomTag(a4),a0
    move.l  a0,RT_NAME(a4)      ; Nom pour FindResident()
    lea ResCode-RomTag(a4),a0
    move.l  a0,RT_INIT(a4)      ; Routine Init

    lea TagPtrs-RomTag(a4),a0  ; Insère notre RomTag
    move.l  a4,(a0)
    move.l  KickTagPtr(a6),4(a0)    ; dans ExecBase.KickTagPtr
    beq.s   .1
    bset    #7,4(a0)    ; Positionne le bit 31 bu mot long
.1  move.l  a0,KickTagPtr(a6)

    lea Modules-RomTag(a4),a0  ; Insère notre module
    move.l  a4,(a0)
    move.l  ResModules(a6),4(a0)    ; dans ExecBase.ResModules
    beq.s   .2
    bset    #7,4(a0)    ; Positionne le bit 31 du mot long
.2  move.l  a0,ResModules(a6)

    lea memList-RomTag(a4),a0  ; Initialise et insère
    move.l  a4,ML_ME+ME_ADDR(a0)    ; notre MemList dans
    move.l  KickMemPtr(a6),LN_SUCC(a0) ; ExecBase
    move.l  a0,KickMemPtr(a6)

    CALL    SumKickData
    move.l  d0,KickCheckSum(a6)     ; checksum

    CALL    Enable

    lea ok_inst.txt(pc),a0 ; Et c'est tout !
    bsr Print
    bra Exit

; ************************************
RemoveTag:
    bsr.s   FindMe
    bne.s   RemoveIt
    lea pas_la.txt(pc),a0
    bsr.s   Print
    bra Exit

RemoveIt:
    movea.l d0,a4       ; a4 pointe le RomTag

    CALL    Disable

    lea memList-RomTag(a4),a1  ; Recherche NOTRE MemList...
    lea KickMemPtr(a6),a0
.loop   move.l  LN_SUCC(a0),d0
    cmpa.l  d0,a1
    beq.s   .found
    movea.l d0,a0
    bra.s   .loop

.found  move.l  LN_SUCC(a1),LN_SUCC(a0)    ; ...et l'enlève de ExecBase

    move.l  TagPtrs-RomTag+4(a4),KickTagPtr(a6)
    bclr    #7,KickTagPtr(a6)  ; Enlève notre KickTagPtr

    move.l  Modules-RomTag+4(a4),ResModules(a6)
    bclr    #7,ResModules(a6)  ; Enlève notre ResModule

    CALL    SumKickData
    move.l  d0,KickCheckSum(a6)     ; checkum

    CALL    Enable

    movea.l BODYadr-RomTag(a4),a1  ; Libère la mémoire du
    move.l  BODYlen-RomTag(a4),d0  ; BODY de l'image IFF
    CALL    FreeMem

    movea.l a4,a1
    move.l  #RESSIZE,d0     ; Libère la mémoire des
    CALL    FreeMem         ; données résidentes

    lea ok_rem.txt(pc),a0  ; Et c'est tout !
    bsr.s   Print
    bra Exit

; ************************************
FindMe  lea TagName(pc),a1
    movea.l $4.w,a6
    CALL    FindResident
    tst.l   d0
    rts

; ************************************
Print   move.l  a6,-(sp)
    moveq   #0,d3
    move.b  (a0)+,d3
    move.l  a0,d2
    move.l  stdout(a5),d1
    movea.l DosBase(a5),a6
    CALL    Write
    movea.l (sp)+,a6
    rts
```

```
; ************************************
    include "LoadPic.s" ; Routine de chargement IFF

; ************************************
VARS    dcb.b   VARSIZE
dosname dc.b    "dos.library",0
    even

usage.txt   dc.b    .1-*-1
        dc.b    "NewPic V 1.0 -  1991, Max pour ANT",10
        dc.b    "Usage : NewPic [I=INSTALL|R=REMOVE]",10
.1      even

deja_la.txt DEFBSTR <"NewPic est déjà installé.">
pas_la.txt  DEFBSTR <"NewPic n'est pas installé.">
memoire.txt DEFBSTR <"Pas assez de mémoire !">
ok_inst.txt DEFBSTR <"NewPic v1.0 installé.">
ok_rem.txt  DEFBSTR <"NewPic v1.0 dé-installé.">

erreurs dc.l    fichier.txt,form.txt,ilbm.txt,reading.txt
fichier.txt DEFBSTR <"Fichier non trouvé.">
form.txt    DEFBSTR <"Ce n'est pas un fichier IFF.">
ilbm.txt    DEFBSTR <"Ce n'est pas un fichier ILBM.">
reading.txt DEFBSTR <"Erreur en lecture du fichier IFF.">

; ************************************
    include "Resident.s"    ; Données et code résident

; ************************************
    END
```

LISTING 2

```
; LoadPic.s - Routine de chargement de l'image IFF

LoadPic movem.l d2-d7/a2-a6,-(sp)
    movea.l DosBase(a5),a6
    lea readbuf(a5),a4

    moveq   #E_FILE,d7  ; Ouvre le fichier
    move.l  a0,d1
    move.l  #MODE_OLDFILE,d2
    CALL    Open
    move.l  d0,d4
    beq nofile

    moveq   #E_IFF,d7
    move.l  d4,d1
    move.l  a4,d2
    moveq   #12,d3
    CALL    Read
    cmpi.l  #'FORM',(a4)    ; Vérifie le 'FORM'
    bne noiff

    moveq   #E_ILBM,d7
    cmpi.l  #'ILBM',8(a4)   ; Vérifie le 'ILBM'
    bne noiff

iff move.l  d4,d1
    move.l  a4,d2
    moveq   #8,d3
    CALL    Read
    subq.l  #8,d0       ; Réellement 8 octets lus ?
    bne loaded
    movem.l (a4),d5-d6  ; d5=ChunkName, d6=ChunkSize

    cmpi.l  #'BMHD',d5
    beq.s   bmhd
    cmpi.l  #'CAMG',d5
    beq.s   camg
    cmpi.l  #'CMAP',d5
    beq.s   cmap
    cmpi.l  #'BODY',d5
    beq.s   body

seek    move.l  d4,d1       ; Saute les Chunks inconnus
    move.l  d6,d2       ; (d6 = longueur du chunk)
    moveq   #OFFSET_CURRENT,d3
    CALL    Seek
    bra.s   iff

bmhd    bsr LoadChunk   ; Charge le chunk en mémoire
    lea width(pc),a0
    move.w  0(a3),0(a0) ; Largeur de l'image
    move.w  2(a3),2(a0) ; Hauteur de l'image
    move.b  8(a3),4+1(a0)   ; Nb plans
    move.b  10(a3),8(a0)    ; Compression
    bsr FreeChunk   ; Libère le chunk
    bset    #16,d7      ; Flag BMHD ok
    bra.s   iff     ; Chunk suivant

camg    bsr.s   LoadChunk   ; Charge le chunk en mémoire
    lea modes(pc),a0
    move.w  2(a3),(a0)  ; ViewModes
    bsr FreeChunk   ; Libère le chunk
    bset    #17,d7      ; Flag CAMG ok
    bra.s   iff     ; Chunk suivant

cmap    bsr.s   LoadChunk   ; Charge le chunk en mémoire
    movea.l a3,a0
    lea palette(pc),a1 ; Buffer de destination
    move.l  d6,d0
    divu    #3,d0
    subq.w  #1,d0       ; d0 = nb. couleurs - 1
    cmpi.w  #31,d0
    ble.s   .loop
    moveq   #31,d0      ; 32 couleurs maximum !
.loop   moveq   #0,d1
    move.b  (a0)+,d1    ; Composante Rouge
    lsl.w   #4,d1
    or.b    (a0)+,d1    ; Composante Verte
    lsl.w   #4,d1
    or.b    (a0)+,d1    ; Composante Bleue
    lsr.w   #4,d1
    move.w  d1,(a1)+    ; La couleur est dans la palette
    dbra    d0,.loop    ; Couleur suivante
    bsr.s   FreeChunk   ; Libère le chunk
    bset    #18,d7      ; Flag CMAP ok
    bra iff     ; Chunk suivant

body    bsr.s   LoadChunk   ; Charge le chunk en mémoire
    lea BODYadr(pc),a0
    move.l  a3,(a0)+    ; Adresse et taille dans la MemList
    move.l  d6,(a0)     ; (qui restera résidente)
    bset    #19,d7      ; Flag BODY ok
    bra iff     ; ON NE LIBERE PAS LE CHUNK !!!

loaded  moveq   #E_NOERR,d6
    swap    d7
    cmpi.w  #%1111,d7   ; Tous les chunks chargés ?
    beq.s   .iffok
    moveq   #E_BAD,d6   ; Non, y'a une erreur...
.iffok  exg d7,d6

noiff   move.l  d4,d1       ; Ferme le fichier
    CALL    Close

nofile  move.l  d7,d0       ; d0 = code d'erreur
    movem.l (sp)+,d2-d7/a2-a6
    rts

LoadChunk:
    movea.l $4.w,a6
    move.l  d6,d0       ; d0 = d6 = sizeof(Chunk)
    moveq   #MEMF_CHIP,d1   ; En CHIP si ça doit rester résident
    CALL    AllocMem    ; Allocation mémoire
    movea.l d0,a3       ; a3 = adresse du Chunk
    move.l  d4,d1
    move.l  d6,d3
    movea.l DosBase(a5),a6
    CALL    Read        ; et lecture du chunk
    sub.l   d6,d0       ; Z=1 si erreur
    rts

FreeChunk:
    movea.l $4.w,a6
    movea.l a3,a1       ; a1 = a3 = adresse du chunk
    move.l  d6,d0       ; d0 = d6 = taille du chunk
    CALL    FreeMem     ; Libération mémoire
    movea.l DosBase(a5),a6 ; (DosBase sert encore !)
    rts
```

LISTING 3

```
; Resident.s - Données et code résident (initialisés ailleurs)

RomTag  dc.w    RTC_MATCHWORD  ; = $4afc
    dc.l    0,0     ; MatchTag, EndSkip
    dc.b    RTF_COLDSTART  ; Flags
    dc.b    1,0,-59     ; Version, Type, Pri
    dc.l    0,0,0       ; Name, IDString, Init
TagName dc.b    "NewPic 1.0",0
    even

memList dcb.b   LN_SIZE
    dc.w    2       ; ML_NUMENTRIES
    dc.l    0,RESSIZE   ; ML_ADR, ML_LENGTH
BODYadr dc.l    0
BODYlen dc.l    0

TagPtrs dc.l    0,0
Modules dc.l    0,0

width   ds.w    1
height  ds.w    1
depth   ds.w    1
modes   ds.w    1
compr   ds.w    1
palette ds.w    32

tdname  TD_NAME
gfxname dc.b    "graphics.library",0
    even

ResCode movem.l d0-d7/a0-a6,-(sp)
    movea.l $4.w,a6

    move.l  #mm_SIZE,d0 ; Alloue les variables dynamiques
    move.l  #MEMF_PUBLIC|MEMF_CLEAR,d1
    CALL    AllocMem
    tst.l   d0
    beq NoVars      ; Raté (ce serait étonnant !)
    movea.l d0,a5       ; a5 pointe les vatriables

    lea mm_port(a5),a2 ; Initialise le MsgPort
    move.b  #NT_MSGPORT,LN_TYPE(a2)
    moveq   #-1,d0
    CALL    AllocSignal
    move.b  d0,MP_SIGBIT(a2)
    bmi NoSig
    suba.l  a1,a1
    CALL    FindTask
    move.l  d0,MP_SIGTASK(a2)

    lea mm_tdio(a5),a1 ; Initialise l'IOExtTD
    move.b  #NT_MESSAGE,IO+MP+LN_TYPE(a1)
    move.l  a2,IO+MP+MN_REPLYPORT(a1)

    lea tdname(pc),a0  ; Ouvre le trackdisk.device (DF0:)
    moveq   #0,d0
    moveq   #0,d1
    CALL    OpenDevice
    tst.b   d0
    bne.s   NoTD

    lea mm_tdio(a5),a3
    lea mm_buff(a5),a4

    movea.l a3,a1
    move.w  #TD_CHANGENUM,IO_COMMAND(a1)
    CALL    DoIO
    move.l  IO_ACTUAL(a3),mm_chng(a5)

    bsr.s   ChkDsk      ; Y'a un disk bootable ?
    beq.s   ResExit     ; Oui -> on s'casse

    bsr Image       ; Non -> on affiche l'image

ChkChng btst    #6,$bfe001 ; (on quitte avec la souris)
    beq.s   .oui

    movea.l a3,a1       ; Disque changé ?
    move.w  #TD_CHANGENUM,IO_COMMAND(a1)
    CALL    DoIO
    move.l  IO_ACTUAL(a3),d0
    cmp.l   mm_chng(a5),d0
    beq.s   ChkChng     ; Pas encore

    bsr.s   ChkDsk      ; Le nouveau disque est bootable ?
    bne.s   ChkChng     ; Non -> boucle

.oui    bsr UnloadImg   ; Enlève l'image

ResExit movea.l a3,a1       ; Ferme le trackdisk.device
    CALL    CloseDevice
NoTD    moveq   #0,d0
    move.b  mm_port+MP_SIGBIT(a5),d0
    CALL    FreeSignal
NoSig   movea.l a5,a1
    move.l  #mm_SIZE,d0
    CALL    FreeMem
NoVars  movem.l (sp)+,d0-d7/a0-a6
    rts

ChkDsk  movea.l a3,a1
    move.w  #TD_CHANGESTATE,IO_COMMAND(a1)
    CALL    DoIO
    tst.l   IO_ACTUAL(a3)
    bne.s   .non        ; Pas de disque dans le drive !

    movea.l a3,a1
    move.w  #TD_CHANGENUM,IO_COMMAND(a1)
    CALL    DoIO
    move.l  IO_ACTUAL(a3),mm_chng(a5)

    movea.l a3,a1       ; Démarre le moteur
    move.w  #TD_MOTOR,IO_COMMAND(a1)
    addq.l  #1,IO_LENGTH(a1)
    CALL    DoIO

    movea.l a3,a1       ; ...et lit les 2 secteurs boot
    move.w  #ETD_READ,IO_COMMAND(a1)
    move.l  #2*TD_SECTOR,IO_LENGTH(a1)
    move.l  a4,IO_DATA(a1)
    clr.l   IO_OFFSET(a1)
    move.l  mm_chng(a5),IOTD_COUNT(a1)
    CALL    DoIO
    move.w  d0,-(sp)    ; (sauve le code d'erreur)

    movea.l a3,a1       ; Arrête le moteur
    move.w  #TD_MOTOR,IO_COMMAND(a1)
    clr.l   IO_LENGTH(a1)
    CALL    DoIO

    tst.w   (sp)+       ; Erreur de lecture ?
    bne.s   .non

    movea.l a4,a0
    cmpi.l  #BBNAME_DOS,(a0)    ; Disquette est bootable ?
    bne.s   .non
    moveq   #0,d0
    move.w  #((TD_SECTOR*2)/4)-1,d1    ; Calcule son checksum
.chksum add.l   (a0)+,d0
    bcc.s   .1
    addq.l  #1,d0
.1  dbra    d1,.chksum
    not.l   d0      ; Z = 1 si le disque est bootable
.non    rts

; ************************************
Image   movem.l a2-a6,-(sp) ; Affichage de l'image IFF

    lea gfxname(pc),a1
    moveq   #0,d0
    CALL    OpenLibrary
    move.l  d0,gfxbase(a5)
    beq .failed
    movea.l d0,a6
    move.l  gb_ActiView(a6),mm_oldv(a5)

    lea mm_view(a5),a2 ; a2 pointe le View
    lea mm_vp(a5),a3   ; a3 pointe le ViewPort
    lea mm_bmap(a5),a4 ; a4 pointe la bitmap
```

```
movea.l a2,a1       ; Initialise le View
CALL    InitView
move.l  a3,v_ViewPort(a2)
move.w  modes(pc),v_Modes(a2)

movea.l a3,a0       ; Initialise le ViewPort
CALL    InitVPort
move.w  width(pc),vp_DWidth(a3)
move.w  height(pc),vp_DHeight(a3)
move.w  modes(pc),vp_Modes(a3)
lea mm_rinf(a5),a0
move.l  a0,vp_RasInfo(a3)

move.w  depth(pc),d1
moveq   #1,d0
lsl.l   d1,d0
CALL    GetColorMap
move.l  d0,vp_ColorMap(a3)
beq .failed

movea.l a4,a0       ; Initialise la BitMap
move.w  depth(pc),d0
move.w  width(pc),d1
move.w  height(pc),d2
CALL    InitBitMap

move.w  depth(pc),d2
subq.w  #1,d2
moveq   #0,d3

move.b  compr(pc),d0    ; Si l'image n'était pas compressée,
beq.s   .norast     ; pas besoin d'allouer des bitplanes !

.planes move.w  width(pc),d0
move.w  height(pc),d1
CALL    AllocRaster
move.l  d0,bm_Planes(a4,d3.l)
beq.s   .failed
addq.l  #4,d3
dbra    d2,.planes
bra.s   .initRI

.norast movea.l BODYadr(pc),a0
move.w  width(pc),d0
lsr.w   #3,d0
mulu    height(pc),d0
.noras1 move.l  a0,bm_Planes(a4,d3.l)
adda.l  d0,a0
addq.l  #4,d3
dbra    d2,.noras1

.initRI lea mm_rinf(a5),a0 ; Initialise le RasInfo
move.l  a4,ri_BitMap(a0)

movea.l a2,a0       ; Construit le View
movea.l a3,a1
CALL    MakeVPort
movea.l a2,a1
CALL    MrgCop

move.b  compr(pc),d0    ; Décompresse au besoin l'image
beq.s   .nopack
bsr.s   UnpackIFF

.nopack movea.l a3,a0       ; ...et affiche le résultat
lea palette(pc),a1
moveq   #1,d0
move.w  depth(pc),d1
lsl.l   d1,d0
CALL    LoadRGB4
movea.l a2,a1
CALL    LoadView
CALL    WaitTOF
.failed movem.l (sp)+,a2-a6
rts

UnpackIFF:
movem.l d0-d7/a0-a6,-(sp)   ; Décompactage IFF
move.l  BODYadr(pc),a4
lea mm_bmap(a5),a3
move.w  bm_BytesPerRow(a3),d5
moveq   #0,d0
Lignes  lea bm_Planes(a3),a2
move.w  depth(pc),d1
subq.w  #1,d1
Planes  movea.l (a2)+,a0
move.w  d0,d2
mulu    d5,d2
adda.l  d2,a0
moveq   #0,d2
Comp    moveq   #0,d3
move.b  (a4)+,d3
bmi.s   Crunch
ext.w   d3
add.w   d3,d2
addq.w  #1,d2
CompL   move.b  (a4)+,(a0)+
dbra    d3,CompL
bra.s   NextByte
Crunch  cmpi.b  #$80,d3
beq.s   NextByte
neg.b   d3
ext.w   d3
add.w   d3,d2
addq.w  #1,d2
move.b  (a4)+,d4
CrunchL move.b  d4,(a0)+
dbra    d3,CrunchL
NextByte:
cmp.w   d5,d2
blt.s   Comp
NextPlane:
dbra    d1,Planes
addq.w  #1,d0
cmp.w   bm_Rows(a3),d0
blt.s   Lignes
movem.l (sp)+,d0-d7/a0-a6
rts

UnloadImg:
move.l  gfxbase(a5),d0 ; Libère la mémoire utilisée
beq.s   .nogfx
movea.l d0,a6
movea.l mm_oldv(a5),a1 ; est-ce vraiment utile ?
CALL    LoadView
CALL    WaitTOF
move.l  mm_vp+vp_ColorMap(a5),d0
beq.s   .1
movea.l d0,a0
CALL    FreeColorMap
move.b  compr(pc),d0
beq.s   .norast
.1  move.w  depth(pc),d2
subq.w  #1,d2
lea mm_bmap+bm_Planes(a5),a2
.planes move.l  (a2)+,d0
beq.s   .2
movea.l d0,a0
move.w  width(pc),d0
move.w  height(pc),d1
CALL    FreeRaster
.2  dbra    d2,.planes
.norast lea mm_vp(a5),a0
tst.l   vp_DspIns(a0)
beq.s   .3
CALL    FreeVPortCopLists
.3  move.l  mm_view+v_LOFCprList(a5),d0
beq.s   .4
movea.l d0,a0
CALL    FreeCprList
.4  move.l  mm_view+v_SHFCprList(a5),d0
beq.s   .5
movea.l d0,a0
CALL    FreeCprList
.5  movea.l a6,a1
movea.l $4.w,a6
CALL    CloseLibrary
.nogfx  rts

RESSIZE EQU *-RomTag    ; Taille des données résidentes
```

par MAX

SubWAY I : LES LAYERS

Si l'on essayait de donner une définition la plus vaste possible des Layers, on pourrait simplement dire que ce sont l'âme des fenêtres d'Intuition. Mais que sont-ils donc exactement et surtout, à quoi servent-ils ?

D'après le dictionnaire anglais-français qui traine toujours sur un coin de moin bureau, le mot "Layer" signifie "couche". Rien à voir cependant avec Pampers, même si le but est identique : éviter les fuites... Car en effet, les Layers sont le coeur même de tout le système de clipping du système graphique de l'Amiga. Excusez du peu.

KESAKO ?

Pour citer ce momument de bêtise qu'est Le Livre du Graphisme sur Amiga (editions Micro-Application, bien sûr), qui pour une fois disait quelque chose d'intelligent : "de même qu'un écran n'est rien d'autre qu'un ViewPort étendu, une fenêtre n'est rien d'autre qu'un Layer étendu. Les Layers se chargent du plus gros du travail nécessité par la gestion des fenêtres".

En fait et pour clarifier les choses, les layers sont des portions rectangulaires d'une même BitMap ; toute sortie graphique (dessin, texte...) dans un layer est limitée aux dimensions de ce layer. Les layers peuvent être déplacés, agrandis ou rétrécis, passés les uns derrière les autres, etc. Exactement comme les fenêtres d'Intuition, en fait.

Il peut arriver qu'une opération mettant en action un layer, révèle un autre layer qui était caché sous lui, ou au contraire en dissimule une partie. Dans ce cas, un flag est positionné pour le layer concerné, qui indique à l'application qu'il est temps de rafraîchir le contenu de ce layer. Nous retrouvons là, encore une fois, la base des messages IDCMP de type REFRESHWINDOW d'Intuition.

UTILISATION DES LAYERS

Les layers sont à ce point important pour le système, qu'une bibliothèque particulière leur est dédiée. Il s'agit bien sûr de la layers.library, que l'on ouvre de manière tout-à-fait conventionnelle :

```
struct LayersBase *LayersBase;

if ((LayersBase=(struct LayersBase *)
    OpenLibrary("layers.library", 0L)) == NULL)
        exit(RETURN_FAIL);
```

Cette bibliothèque se charge de créer les layers, de les manipuler (déplacement, changement de taille, superposition, etc.), mais n'y effectue aucune sortie graphique. C'est bien entendu la graphique.library qui s'en charge, ce qui explique que ces deux bibliothèques soient souvent utilisées de concert. De même, tous les calculs inhérants au clipping sont gérées par la graphics.library, la layers.library ne se chargeant que de la mise en place des zones de clipping à l'intérieur des layers. Nous aurons l'occasion d'en reparler un peu plus loin.

Lorsque l'on crée un layer à l'aide de la fonction adéquate, un RastPort lui est automatiquement associé, qui permet l'utilisation des primitives graphiques dans ce layer. Au risque de me répéter, toutes les opérations graphiques dans ce RastPort sont alors clippées à ses limites. A l'inverse, lorsque l'on crée un RastPort directement (à l'aide de la fonction InitRastPort() par exemple), aucun layer ne lui est encore associé, et les sorties graphiques ne sont pas clippées.

Tous les layers se partageant une même BitMap, se partagent également une structure Layer_Info qui aide le système à ne pas se mélanger les pédales dans tout ce foutoir (car c'en est un, croyez-moi !).

Il existe également une autre notion importante pour le clipping : les Regions. Il s'agit de portions rectangulaires des layers, qui sont utilisées pour clipper les sorties graphiques à l'intérieur des layers. Cette notion peut paraître obscure et inutile ; nous verrons plus loin qu'il n'en n'est rien et quelle est leur utilité réelle.

Enfin, les layers fournissent un mécanisme de verrouillage ("Lock"), basé sur les Semaphores d'Exec, qui empêche provisoirement d'autres tâche d'accéder à ce layer et/ou à la BitMap qu'il partage.

CREATION DE LAYERS

Il existe trois types de layers, que vous reconnaîtrez sûrement si vous êtes un habitué des fenêtres.

- layers Simple Refresh
Toutes les sorties graphiques sont limitées à la (aux) parties visibles du layer. Ce qui "sort" du layer est simplement perdu ; lorsqu'une partie du layer redevient visible, un flag est positionné indiquant que l'application doit redessiner cette partie.

- layers Smart Refresh
Quand une sortie graphique s'effectue dans une partie invisible du layer, ou quand un autre layer vient le recouvrir, le système alloue dynamiquement une mémoire tampon dans laquelle il sauve les informations cachées ; plus tard, lorsque ce layer redeviendra visible, le système copiera cette mémoire tampon dans le layer.

- layers SuperBitMap
Le principe est identique aux layers Smart Refresh, si ce n'est que le système n'alloue aucune mémoire tampon ; c'est au contraire l'application qui fournit une BitMap particulière qui servira de mémoire tampon. Cette BitMap peut être plus grande que le layer lui-même, ce qui permet des scrollings.

De plus, chacun des types de layers ci-dessus peut également être BackDrop, c'est-à-dire qu'il apparaîtra toujours derrière tous les autres layers et ne pourra pas être superposé à d'autres layers.

Les constantes C correspondant à ces différents types de layers sont définies dans graphics/layers.h et se nomment respectivement `LAYERSIMPLE, LAYERSMART, LAYERSUPER et LAYERBACKDROP.`

Pour créer un layer, il faut d'abord disposer une structure Layer_Info. Si vous travaillez avec Intuition, chaque écran possède sa propre structure Layer_Info que vous pouvez utiliser ; sinon, il faudra créer la vôtre au moyen de la fonction NewLayerInfo() :

```
/* Méthode 1 : Récupère le Layer_Info de l'écran courant */
struct Screen *screen;
struct Layer_Info *layer_info;

    if ((Screen = OpenScreen(&newscreen)) != NULL)
        layer_info = &screen->LayerInfo;

/* Méthode 2 : Utilise NewLayerInfo() */
struct Layer_Info *layer_info;

    if ((layer_info = NewLayerInfo()) == NULL)
        clean_exit();
```

NewLayerInfo() alloue une structure Layer_Info et toutes les structures qui lui sont rattachées. Elle initialise de plus certaines paramètres par défaut des layers. Pour libérer cette mémoire, il faudra donc utiliser le pendant de NewLayerInfo(), à savoir DisposeLayerInfo() :

```
/* Libère une Layer_Info allouée par NewLayerInfo() */
    DisposeLayerInfo(layer_info);
```

A partir de là, vous pouvez créer tous les layers que vous voudrez (du moins, tant que la mémoire libre le permettra), en utilisant l'une des

fonctions CreateUpfrontLayer() ou CreateBehindLayer(), suivant que le nouveau layer doit apparaître devant ou derrière tous les autres (layers BackDrop non compris). Ces deux fonctions s'utilisent exactement de la même manière.

CreateUpfrontLayer(li,bm,x1,y1,x2,y2,t,sb) ?
CreateBehindLayer(li,bm,x1,y1,x2,y2,t,sb))

Les paramètres à ces deux fonctions sont :
- *li :* pointeur sur la structure Layer_Info commune à tous les layers ;
- *bm :* pointeur sur la BitMap commune à tous les layers ;
- *x1, y1, x2, y2 :* coordonnées du layer ;
- *t :* type du layer ;
- *sb :* pointeur sur une BitMap supplémentaire, pour le cas où le layer est de type LAYERSUPER ; NULL sinon.

Ces fonctions renvoient NULL si la création du layer a échoué (généralement par manque de mémoire).

Une fois le layer créé, vous pouvez récupérer un pointeur sur son RastPort directement dans la structure Layer :

```
struct RastPort *rp;
rp = layer->rp;
```

ATTENTION : la structure Layer_Info est définie dans graphics/layers.h ; la structure Layer est définie dans graphics/clip.h.

Une fois tous vos layers créés, vous pourrez les déplacer, les agrandir, changement leur ordre de superposition, faire défiler leur contenu (pour les layers de type LAYERSUPER uniquement) et déterminer quel layer se trouve actuellement à une coordonnée précise grâce à la batterie de fonctions que voici (note : dans ces fonctions, le paramètre 'dummy' est pour l'instant inutilisé met doit être mis à 0 pour permettre la compatibilité avec de futures version de la bibliothèque) :

`MoveLayer(dummy, layer, dx, dy)`
Déplace le layer spécifié de (dx, dy) pixels.

`SizeLayer(dummy, layer, dx, dy)`
Agrandit/rétrécit le layer spécifié de (dx, dy) pixels.

`BehindLayer(dummy, layer)`
Place le layer spécifié derrière tous les autres, à l'exception des layers BackDrop.

`UpfrontLayer(dummy, layer)`
Place le layer spécifié devant tous les autres, sauf s'il s'agit d'un layer BackDrop.

`MoveLayerInFrontOf(layer1, layer2)`
Place le layer1 devant le layer2, sauf si le layer1 est du type BackDrop.

`ScrollLayer(dummy, layer, dx, dy)`
Déplace le contenu du SuperLayer spécifié de (dx, dy) pixels. Il s'agit d'un scrolling "soft", effectué au Blitter.

`WhichLayer(layerinfo, x, y)`
Retourne un pointeur sur le layer se trouvant exactement sous les cordonnées (x, y) spécifiées.

`DeleteLayer(dummy, layer)`
Supprime le layer spécifié du système.

Nous continuerons ce passage en revue des layers dans notre prochain numéro. D'ici là, amusez-vous avec le petit programme que je vous ai concocté ici ; il est assez difficile de faire un programme réellement démonstratif des capacités des layers, aussi celui-ci se contente-t'il de créer un layer sur l'écran du Workbench et de le déplacer à volonté. Le mois prochain, vous aurez droit à un programme plus conséquent, n'utilisant pas d'écran Intuition et mettant en oeuvre un layer de type SuperBitMap.

Max
en tient une sacrée couche

```
#include <stdlib.h>
#include <exec/types.h>
#include <intuition/screens.h>
#include <graphics/clip.h>
#include <graphics/layers.h>

#include <proto/exec.h>
#include <proto/intuition.h>
#include <proto/graphics.h>
#include <proto/layers.h>

#define DUMMY 0L

/* Variables globales */
struct Library *GfxBase, *LayersBase, *IntuitionBase;

/* Prototypes */
void main(void);
void cleanExit(void);
void layer_demo(struct Layer *lay);

/* Ouverture des bibliothèques */
void main(void)
{
struct BitMap *bm;
struct RastPort *rp;
struct Layer_Info *li;
struct Layer *lay;
struct Screen WBScreen;

    if (!(IntuitionBase = OpenLibrary("intuition.library", 33L)))
        cleanExit();

    if (!(GfxBase = OpenLibrary("graphics.library", 33L)))
        cleanExit();

    if (!(LayersBase = OpenLibrary("layers.library", 0L)))
        cleanExit();

    GetScreenData( (UBYTE *)&WBScreen, sizeof(WBScreen),
                   WBENCHSCREEN, NULL);
    li = &WBScreen.LayerInfo;
    bm = &WBScreen.BitMap;

    if (!(lay = CreateUpfrontLayer(li, bm,
                                   0, 0, 199, 99,
                                   LAYERSMART, NULL)))
        cleanExit();

    rp = lay->rp;

    /*** Rend le layer visible et affiche un texte dedans */
    SetRast(rp, 2);   SetDrMd(rp, JAM1);
    Move(rp, 10, 10); Text(rp, "Et un Layer, un !", 17);
    Move(rp, 10, 20); Text(rp, "C'est une gomme !", 17);

    /*** Démo layer */
    layer_demo(lay);

    /*** Et c'est fini */
    DeleteLayer(DUMMY, lay);

    cleanExit();
}

/* Fermeture des bibliothèques, retour au CLI */
void cleanExit(void)
{
    if (LayersBase)    CloseLibrary(LayersBase);
    if (GfxBase)       CloseLibrary(GfxBase);
    if (IntuitionBase) CloseLibrary(IntuitionBase);
    exit(0);
}

/* Démo layer (déplacement...) */
void layer_demo(struct Layer *lay)
{
register int ii;

    /*** Déplace le layer sur l'écran du WB (et l'efface !) */
    for (ii = 0; ii < 44; ii++) MoveLayer(DUMMY, lay,  10,   0);
    for (ii = 0; ii < 15; ii++)   MoveLayer(DUMMY, lay,   0,  10);
    for (ii = 0; ii < 44; ii++)   MoveLayer(DUMMY, lay, -10,   0);
    for (ii = 0; ii < 15; ii++)   MoveLayer(DUMMY, lay,   0, -10);
}
```

Par Max

HARDWARE : TOUCHE PAS A

De toutes les interfaces de l'Amiga, le clavier est sans aucun doute la plus utilisée, juste après la souris. Pourtant, peu de programmeurs savent comment il fonctionne et comment l'utiliser efficacement.

Avec un peu de bonne volonté, on peut facilement réaliser une routine de gestion du clavier qui soit en total accord avec les conseils des créateurs de l'Amiga. Nombre de démos et de jeux utilisent des routines toutes faites dont le seul travail consiste à lire un octet à une adresse donnée, pour savoir quelle touche a été appuyée. Ce procédé fonctionne et a le mérite d'être simple. Il n'est toutefois pas très sûr et l'on peut "manquer" plusieurs touches sans même s'en rendre compte.

GENERALITES

Le clavier de l'Amiga a 96 touches et fait partie de cette catégorie de claviers dits "intelligents", car il possède son propre micro-processeur, en l'occurence un 6500/1, de la famille du 6502. Il s'agit d'un micro-processeur 8 bits disposant de sa propre Rom (2 Ko) qui décharge le 68000 de tout le travail fastidieux de reconnaissance de la ou des touches appuyées. Le processeur central reste toutefois obligé de capturer cette touche dans un buffer et de transmettre l'information adéquate aux différentes applications concernées (en l'occurence, le keyboard.device, qui lui-même transmet l'information à l'input.device, qui lui-même la transmet à Intuition et au console.device. Ouf !).

Le clavier émet deux signaux, l'un unidirectionnel (du clavier vers l'Amiga), l'autre bidirectionnel (communication possible dans les deux sens) : il s'agit de KDAT et de KCLK. KDAT est relié au port série du CIAA-A (registre SP, adresse $BFEC01) ; c'est à travers lui que le 68000 pourra lire le code clavier envoyé). KCLK est relié à la broche CNT de ce même CIA (registre ICR, adresse $BFEE01, bit 6). Revoyez au besoin mes articles sur les CIAs dans les ANT 22 et 23.

LE TRANSFERT DES DONNEES

Le schéma de communication est alors le suivant : le clavier met KLCK et KDAT à l'état haut (rappel : ces deux signaux sont actifs à l'état bas). Il commence la transmission en mettant le premier bit de donnée sur KDAT, suivi d'une pulsation sur KLCK (bas puis haut). Le second bit de donnée est alors envoyé, suivi d'une nouvelle pulsation, et ainsi de suite jusqu'au huitième bit (pour information, la transmission d'un bit dure exactement 60 microsecondes, pulsation comprise ; de fait, la vitesse de transmission est de 17 kbits par seconde). Lorsque tous les bits ont été envoyés et la dernière pulsation émise, le clavier remet KDAT à l'état haut. Ce n'est également que lorsque les huit bits ont été reçus, c'est-à-dire lorsque son port série est plein, que le CIA-A déclenche une interruption de niveau 2. A partir de là, c'est au 68000 de jouer.

La première chose que le micro-processeur doit faire (juste après avoir lu le caractère, bien sûr), c'est signaler au clavier la bonne réception de ce caractère. Cela se fait en mettant KDAT à l'état bas pendant au mois 1 microseconde, puis de nouveau à l'état haut. Si l'on en croit les recommendations du Hardware Manual, 1 microseconde est un délai minimal que seuls les meilleurs claviers (entendez par "là ceux disposant des meilleurs composants") pourront détecter ; le logiciel doit émettre le signal de handshake pendant au moins 85 microsecondes pour être totalement compatible avec tous les modèles de clavier passés, présents ou futurs.

Le clavier attend ce signal de handshaking pendant un maximum de 145 millisecondes. Si passé ce délai il n'a pas reçu le signal, il considère que la transmission a raté et entre dans un mode spécial de correction d'erreur. Ce mode consiste à émettre des 1 à espaces réguliers de 145 millisecondes sur KDAT jusqu'à ce qu'enfin, le signal arrive. Une fois la ligne rétablie, un faux-caractère aura été transmis à l'ordinateur, qui n'aura entre temps reçu que des 1 (nous verrons un tout petit peu plus loin que celà ne prête absolument pas à conséquence et comment les ingénieurs de chez Commodore-Amiga ont résolu le problème). Le clavier envoie alors un code spécial ($F9) à l'Amiga pour lui signaler que le précédent caractère était erroné et que le même va être retransmis. Notez que ce code n'aide en rien à la resynchonisation des deux processeurs ; il avertit seulement le logiciel système qu'une erreur est survenue. Le clavier pourrait tout aussi bien retransmettre le caractère fautif sans rien dire à personne.

LE FORMAT DES DONNEES

Nous l'avons vu, les données du clavier sont émises en série vers le CIA-A. Les codes clavier sont codés sur 7 bits seulement (valeurs de $00 et $7F, soit de 0 à 127 en décimal), ce qui est amplement suffisant. Le huitième bit est employé comme drapeau pour indiquer si la touche concernée a été enfoncée (0) ou relâchée (1). En effet, lorsque l'on appuie sur une touche, par exemple une lettre, celà suppose deux actions : d'abord l'enfoncer, puis la relâcher. Le clavier distingue ces deux actions afin de limiter le nombre de transferts vers l'ordinateur lorsqu'une touche est maintenue enfoncée pendant longtemps. Dans tous les cas, les bits 7 à 1 de l'octet reçu par le CIA contiendront le code clavier effectif et le bit 0, le drapeau KEY UP/KEY DOWN. Quand le 68000 aura lu cet octet sur le port série, il devra donc lui faire subir une rotation d'un bit vers la droite (à l'aide de l'instruction ROR) afin de reconstituer le véritable code-clavier. Le drapeau se retrouvera quant à lui dans le bit 7.

Seule la touche Caps-Lock fait exception à cette règle ; le clavier n'envoie de code la concernant que lorsqu'elle a été enfoncée. Au premier appui sur cette touche, le clavier envoie son code avec le drapeau KEY UP à 0. L'Amiga sait donc que cette touche vient d'être activée (entre temps, le clavier aura également allumé la LED de la touche). Au second appui sur Caps-Lock, le clavier envoie son code avec cette fois-ci le drapeau KEY UP à 1. La LED s'éteint et l'Amiga sait que Caps-Lock n'est plus active. En celà, Caps-Lock agit exactement comme si l'on maintenait une touche Shift enfoncée pendant longtemps. Ce qui serait évidemment bien moins pratique.

La raison du pré-décalage du code caractère par le clavier réside justement dans le protocole de correction d'erreur vu plus haut : dans ce mode, le clavier n'envoie que des 1 à l'Amiga, jusqu'à ce qu'il reçoive le signal de handshaking. De fait, lorsque la communication sera rétablie, l'Amiga croira avoir à faire à une touche relâchée (le dernier bit étant à 1, vous suivez encore ?), ce qui est beaucoup moins grave que s'il pensait avoir à faire à une touche enfoncée.

N'oubliez pas enfin que la ligne KDAT est active à l'état bas, c'est-à-dire qu'un 1 reçu correspond en fait à un 0 et inversemment. Pour palier à celà, le 68000 devra inverser chaque bit du code reçu avant de réellement pouvoir l'exploiter. Ceci est réalisé très simplement par une simple instruction NOT.

CODES SPECIAUX

Le clavier transmet quelques code spéciaux à l'Amiga, qui ne sont pas des numéros de touches, pour lui signaler un état particulier. Nous en avons déjà vu un exemple avec le code $F9, qui signifie que la synchronisation entre les deux processeurs a été perdue. Il en existe d'autres, que voici réunis dans le tableau ci-dessous.

```
Code    Signification
$78 Avertissement Reset. Le clavier signale ainsi à l'Amiga
    que les touches CTRL-Amiga gauche-Amiga droite ont été
    pressées simultanément.
```

```
$F9 Erreur de synchronisation.
$FA Le buffer de caractères du clavier est plein.
$FB - inutilisé -
$FC L'auto-test du clavier a échoué.
$FD Début de séquence PowerUp. Le clavier transmet à l'Amiga
    tous les code claviers qu'il a enregistré pendant son
    initialisation, et qu'il ne pouvait donc pas encore
    émettre.
$FE Fin de séquence PowerUp.
$FF - inutilisé -
```

Le code $78 n'est malheureusement plus en vigueur depuis longtemps. Sur les premiers A1000, il était envoyé à l'Amiga pour lui signaler que l'utilisateur avait demandé un Reset. Après s'être acquitté d'un protocole de réponse particulier, l'Amiga disposait alors de 10 secondes de répit pour se préparer au Reset, c'est-à-dire principalement terminer tous les accès disques en cours. Au terme de ces 10 secondes, le clavier ré-initialisait l'ordinateur. Ceci n'est plus possible depuis l'apparition des A500 et A2000.

Les codes $FC, $FD et $FE sont envoyés suivant le résultat de l'auto-test auquel le clavier se soumet à chaque ré-initialisation. Ce test inclue un checksum de sa Rom et de sa Ram (celle qui contient le buffer de caractères et les variables de son programme) et une vérification de son timer. Une fois ce test effectué, il essaye d'entrer en synchronisation avec l'Amiga suivant la procédure décrite plus haut. Ce n'est qu'une fois synchronisé qu'il envoie à l'Amiga le résultat de son test. Si celui-ci a été négatif, le code $FC est envoyé et le clavier fait clignoter la LED de la touche Caps-Lock suivant un protocole défini ainsi :

```
Clign.  Cause de l'erreur
1 fois  Le checksum de la Rom est incorrect
2 fois  Le checksum de la Ram est incorrect
3 fois  Le timer ne fonctionne pas correctement
```

Après ceci, le clavier tente de se ré-initialiser. L'Amiga quant à lui peut très bien fonctionner sans clavier ; ce code ne sert qu'à l'informer que celui-ci n'est pas en état de travailler.

Si au contraire, son auto-test est positif, le clavier transmet à l'Amiga tous les codes des touches qui auraient pu être appuyées durant son initialisation. Celà commence par le code spécial $FD, suivi des codes bufférisés sans prendre en compte leur relâchement. En d'autres termes, tous ces codes sont envoyés avec le drapeau KEY UP à 0 (touche enfoncée). Finalement, le clavier conlut cette série par un second code spécial, $FE.

Il est très peu probable qu'une touche soit appuyée avant que le clavier n'ait eu le temps de faire son auto-test et de se synchroniser avec l'ordinateur. De fait, la séquence typique de démarrage du clavier sera (en assumant que tout fonctionne correctement) :

1) initialisation interne
2) synchronisation avec l'ordinateur
3) transmission de $FD
4) transmission de $FE

Dans tous les cas, $FD et $FE sont transmis, même si aucune touche n'a été préssée.

LES CODES CLAVIER

Pour terminer cet article, et avant que d'attaquer à proprement parler la programmation du clavier, je vous propose une double liste des codes claviers que le 6500/1 envoie à l'Amiga. La première résume ces codes par ordre numérique pour vous permettre de savoir à quelle touche correspond quel code ; la seconde, sous forme graphique, vous permettra de repérer rapidement le code d'une touche particulière. Quant à l'invévitable listing, vous devrez patienter un petit mais long mois avant d'en profiter, la place qui me reste étant loin d'être suffisante. Vous avez le droit si le coeur vous en dit d'écrire massivement à la rédaction pour demander plus de pages pour la rubrique Hardware. Adressez votre courrier à Stéphane Schreiber.

Codes de $00 à $3F

Ces codes sont ceux assignés aux touches aplhanumériques et de ponctuation. Les lettres représentées sur les touches sont bien entendu différentes de pays en pays, mais les codes clavier restent les mêmes. Les touches $2B et $30 sont des exceptions ; elles n'existent pas sur le clavier original américain et ont été ajoutées pour balancer le nombre plus élevé de caractères sur les claviers européens.

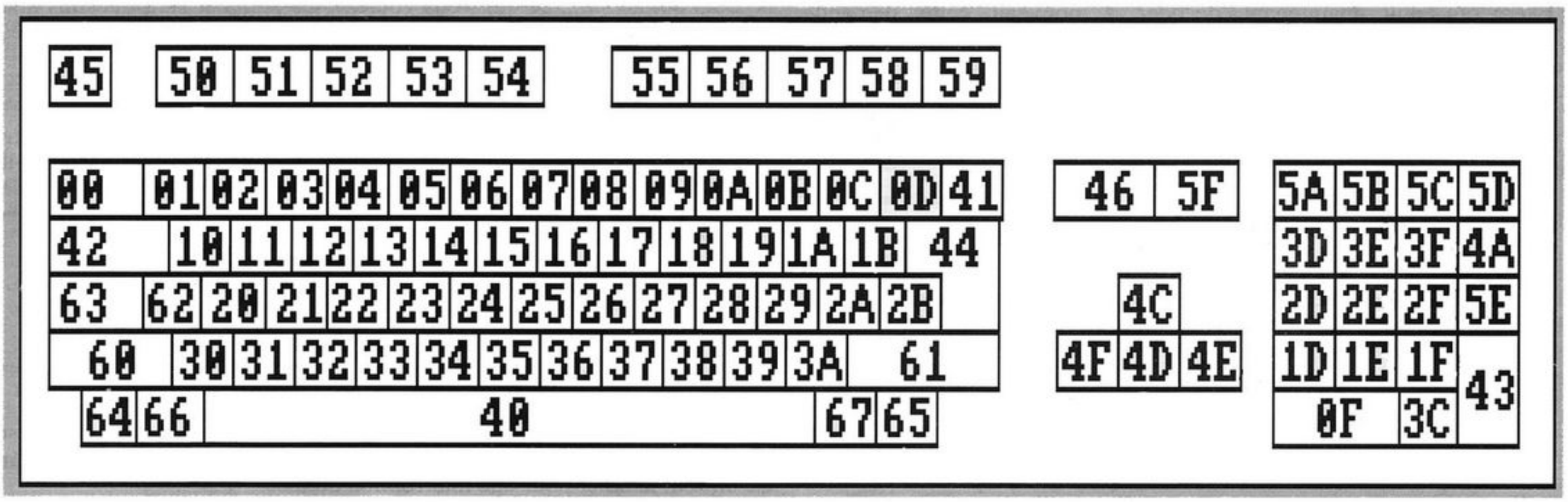
Codes de $40 à $5F

Ces touches sont communes à tous les types de claviers.

```
40  Espace
41  BackSpace (retour arrière)
42  Tab
43  Entrée (pavé numérique)
44  Return (pavé principal)
45  Esc
46  Del
4C  Curseur haut
4D  Curseur bas
4E  Curseur droit
4F  Curseur gauche
50-59   Touches de fonctions F1 à F10
5F  Help
```

Codes de $60 à $67

```
60  Shift de gauche
61  Shift de droite
62  Caps-Lock
63  Control
64  Alt de gauche
65  Alt de droite
66  Amiga de gauche (ou Commodore)
67  Amiga de droite
```



ExecBase II : LES INTERRUPTIO

L'Amiga utilise de manière peu orthodoxe les 7 niveaux d'interruption du 68000. Grâce à Paula, il est capable de délivrer pas moins de 15 sources d'interruptions différentes, classées par priorité.

Comme on était en droit de s'y attendre, Exec tire pleinement parti de cette particularité du hardware, à tel point que les interruptions sont la base même de son noyau multitâche. Exec décode les demandes d'interruptions, les évalue et se branche sur les routines adéquates, en fonction d'un certain nombre de paramètres que vous allons voir ici. En plus des niveaux de priorité, Exec distingue deux types d'interruptions : les interruptions en provenance du hardware, bien sûr, mais aussi celles provoquées par logiciel. Enfin, Exec permet l'inhibition complète et la restitution de toutes les interruptions sur simple demande d'une application.

COMMENT CA MARCHE

Avant qu'une routine d'interruption ne soit appelée, plusieurs étapes, tant hardwares que logicielles, sont exécutées. La séquence exacte de ce qui se passe est la suivante :
1) un périphérique quelconque décide de causer une interruption et envoie pour ce faire un signal à Paula.
2) Paula prend en compte cette demande et la "note" en positionnant le bit correspondant du registre hardware INTREQ. Puis elle examine le bit correspondant dans INTENA pour déterminer si cette interruption est autorisée ou non. Si et seulement si c'est le cas, Paula la transmet au 68000, non sans avoir auparavant transformé le niveau de priorité en un acceptable par le micro-processeur.
3) le 68000 décode à son tour cette demande d'interruption et détermine si elle est valide ou non. Une interruption n'est valide que lorsque son niveau de priorité est supérieur à celui dans lequel se trouve le micro-processeur au moment de la demande.
4) si la demande d'interruption est valide, le processeur passe alors en mode superviseur (s'il ne l'était pas déjà), et sauvegarde sur la pile les informations relatives à son état actuel (registres SR et PC), puis il se place dans le même niveau de priorité que celui de l'interruption.
5) le 68000 cherche maintenant dans sa table d'auto-vecteurs d'interruptions, l'adresse de la routine à exécuter (note : sur un 68000, cette table se trouve aux adresses $64 à $7C incluses. Sur un 68020 ou 68030, un registre supplémentaire nommé VBR - pour Vector Base Register - permet de reloger cette table n'importe où dans l'espace d'adressage du processeur).
6) le 68000 saute à l'adresse lue dans la table. Il s'y trouve une routine d'Exec qui doit maintenant décoder à nouveau cette demande d'interruption, afin d'en retrouver l'originaire. Ceci est réalisé au moyen des registres hardware INTREQ et INTENA. Une fois l'origine de l'interruption déterminée, Exec recherche dans la structure ExecBase l'adresse de la routine à exécuter pour ce type d'interruption.
7) Exec appelle finalement cette routine, qui peut, dans le cadre de serveurs d'interruption, en appeler encore d'autres.

Comme on peut le voir, celà est loin d'être simple !

Au retour de chaque interruption, Exec détermine si le temps imparti à la tâche en cours est écoulé ou non, ou si une tâche a reçu un signal qu'elle attendait (via Wait() ou WaitPort()), auquel cas le gestionnaire de tâches est appelé et le temps d'occupation du processeur pour chaque tâche est recalculé.

Comme vous le voyez, Exec repose lourdement sur les interruptions pour gérer son multitâche. Si, pour une raison quelconque, les interruptions ne survenaient plus, la tâche courante utiliserait tout le temps CPU pour elle toute seule, plus rien ne venant la forcer à l'abandonner quelques instants.

LES PRIORITES DES INTERRUPTIONS

Les interruptions sont classées par priorité à deux niveaux : d'abord par le 68000 lui-même, puis par le logiciel qui, à l'intérieur de chaque niveau de priorité du processeur, introduit des pseudo-priorités. Ces pseudo-priorités déterminent l'ordre d'exécution de chaque interruption pour une priorité 68000 donnée.

Il existe en fait deux types de gestionnaires d'interruptions : les handlers, qui en sont les propriétaires exclusifs, et les serveurs, qui se partagent l'interruption.

Le tableau ci-dessous indique pour chaque niveau d'interruption, son type de gestionnaire.

```
Pri CPU Pri Exec    Type     Origine INTREQ
1     1  H   software     SOFTINT
      2  H   transfert disque terminé   DSKBLK
      3  H   buffer série vide  TBE
2     4  S   externe et CIA-A   PORTS
3     5  S   Copper  COPER
      6  S   VBL VERTB
      7  H   Blitter BLIT
4     8  H   voie audio 2    AUD2
      9  H   voie audio 0    AUD0
     10  H   voie audio 3    AUD3
     11  H   voie audio 1    AUD1
5    12  H   buffer série plein RBF
     13  H   disk sync   DSKSYNC
6    14  S   externe et CIA-B    EXTER
     15  -   (pas une interruption) INTEN
7    --  S   interruption non masquable ---
```

La colonne INTREQ indique les noms de bits, tels qu'ils apparaissent dans ce registre.

CONTROLE DES INTERRUPTIONS

Revenons maintenant d'un peu plus près à Exec. Pour mettre en place une interruption, il faut remplir devinez quoi ? Une structure, oui. Elle se nomme Interrupt et est définie ainsi :

```
struct Interrupt
{   struct Node is_Node;
    APTR is_Data;
    VOID (*is_Code)();
};
```

Le noeud (is_Node) permet à Exec de maintenir trié par ordre de priorités décroissantes, toutes les interruptions. Il est de type NT_INTERRUPT.
is_Data pointe sur... ce que vous voulez ! Ce pointeur sera transmis à la routine d'interruption. C'est l'idéal pour partager des variables et des données avec le programme principal.
Enfin, is_Code pointe sur le code de la routine d'interruption elle-même.

Note : très peu de fonctions du système d'exploitation peuvent être appelées depuis une interruption. Notamment, toutes les fonctions faisant appel, directement ou indirectement, au gestionnaire de mémoire d'Exec sont a prohiber. En général, les fonctions suivantes sont utilisables sans danger depuis une interruption :

`Alert(), Disable(), Enable(), Signal(), Cause(), PutMsg(), ReplyMsg(), FindPort(), FindTask()`

LES HANDLERS

Comme on l'a déjà mentionné auparavant, un handler d'interruption est une routine qui gère exclusivement cette interruption. Il ne peut y avoir qu'un seul handler pour une interruption donnée.

NS SOUS EXEC

Un handler est appelé par Exec comme s'il s'agissait d'une sous routine avec une instruction JSR. La dernière instruction du handler doit donc être RTS, et non RTE. En entrée, certains registres du processeurs contiennent des informations pertinentes : ce sont notamment A0, qui pointe sur l'adresse de base des Custom Chips ($DFF000), A1 qui contient le champ is_Data de la structure Interrupt, A5 pointe sur la routine d'interruption elle-même (Exec effectue en effet un JSR (A5)) et A6 pointe sur ExecBase. Les registres D0-D1/A0-A1/A5-A6 sont librement modifiables, les autres doivent absolument être préservés. De plus, il appartient au handler d'effacer la demande d'interruption dans le registre hardware INTREQ.

On installe un handler à l'aide de la fonction SetIntVector(), qui s'utilise comme suit :

```
SetIntVector(ULONG IntNumber, struct Interrupt *interrupt);
```

IntNumber contient le numéro de l'interruption désirée, par exemple INTB_BLIT (voir "hardware/intbits.h" et "hardware/intbits.i"), et interrupt pointe sur votre structure Interrupt, correctement initialisée. Cette fonction retourne un pointeur sur la structure Interrupt du handler précédemment installé, s'il y en avait un. Il ne faudra pas oublier de remettre le vecteur dans son état initial lorsque votre programme se terminera, avec SetIntVector(IntNum, oldHandler).

LES SERVEURS

Contrairement aux handlers, les serveurs d'interruptions se partagent une même interruption. Exec maintient une liste des serveurs déclarés pour chaque interruption, et les appelle tour à tour. Rappelons toutefois que seules les interruptions PORTS, COPER, VERTB, EXTER et NMI suppportent des serveurs ; les autres requièrent obligatoirement un handler.

Avant d'appeler le serveur suivant dans la chaine, Exec teste le flag Z du 68000 ; si ce flag n'est pas positionné, Exec arrête là le traitement de l'interruption. Un moyen simple de positionner le flag Z consiste à utiliser l'instruction MOVEQ :

```
MOVEQ #0,D0 ; Z est mis, la chaîne continue
```

ou

```
MOVEQ #1,D0 ; Z est faux, la chaîne s'arrête
```

Comme pour un handler, un serveur se termine obligatoirement par une instruction RTS.

Par contre, seuls les registres A1 et A5 contiennent une information valide : A1 contient le champ is_Data de la structure Interrupt et A5 pointe sur la routine d'interruption elle-même. Il ne faut pas s'attendre à trouver quoique ce soit de particulier dans A0 ni dans A6 (en fait, seul le premier serveur de la chaîne, celui qui a la plus haute priorité, reçoit effectivement l'adresse des Custom Chips dans A0 et un pointeur sur ExecBase dans A6. Mais ces registres étant librement modifiables par le serveur, rien ne garantit aux suivants qu'ils seront encore valides).

Note : un bug dans le serveur d'interruption VERTB de la graphics.library assume que quoiqu'il arrive, A0 pointe sur les Custom Chips ($DFF000). La priorité de ce serveur est de 10. Donc, si vous ajoutez un serveur VERTB de priorité supérieure à 10, vous devez absolument terminer votre routine par les trois instructions :

```
LEA $DFF000,A0 ; et la graphics.library sera contente...
MOVEQ #0,D0    ; positionne Z : la chaîne continue
RTS            ; retour à Exec
```

Enfin, les registres D0-D1/A0-A1/A5-A6 sont libremenent modifiables. Les autres doivent obligatoirement être préservés.

On installe un serveur d'interruption à l'aide de la fonction AddIntServer(), qui s'utilise comme suit :

```
AddIntServer(ULONG IntNum, struct Interrupt *interrupt);
```

Les paramètres sont les même que pour SetIntVector(), à la différence qu'AddIntServer() ne retourne aucune valeur en réponse. Pour enlever votre serveur de la chaîne en fin de programme, utilisez RemIntServeur(IntNum, interrupt).

VACANCES

Bon, on arrête là le massacre ; je vous laisse le loisir de découvrir un petit programme de mon cru qui initialise et ajoute un serveur VERTB. Le programme principal est en C et la routine d'interruption, en assembleur. On termine avec les interruptions softs le mois prochain.

LISTING 1

```
/************************************************
* IntServ.c (v1.1)                              *
* Met en place un serveur d'interruption VBL    *
*                                               *
* S. Schreiber - 030590                         *
************************************************/

/*
* Includes
*/
#include <stdio.h>
#include <stdlib.h>
#include <exec/types.h>
#include <exec/memory.h>
#include <exec/interrupts.h>
#include <intuition/intuition.h>
#include <intuition/screens.h>
#include <graphics/gfx.h>
#include <hardware/intbits.h>

#include <proto/exec.h>
#include <proto/intuition.h>
#include <proto/graphics.h>

/*
* Defines
*/
#define WIDTH  640 /* Largeur de l'écran */
#define HEIGHT 128 /* Hauteur ''  ''     */
#define DEPTH  1   /* Profondeur  ''     */

/*
* Types
*/
struct IntData
{   PLANEPTR id_Bitplane, id_Current;
    SHORT    id_BplSize,  id_BplCount;
};

/*
* Variables globales
*/
struct IntuitionBase *IntuitionBase = NULL;
struct GfxBase *GfxBase = NULL;
struct Window *win = NULL;
struct Screen *scr = NULL;

/*
* Données globales
*/
struct NewScreen ns =
{   0, HEIGHT, WIDTH, HEIGHT, DEPTH,
    0, 0,
    HIRES, CUSTOMSCREEN|SCREENQUIET,
    NULL, NULL, NULL, NULL
};

struct NewWindow nw =
{   0, 0, 640, 40,
    0, 1,
    CLOSEWINDOW|INTUITICKS,
    WINDOWCLOSE|WINDOWDRAG|WINDOWDEPTH|NOCAREREFRESH|ACTIVATE,
    NULL, NULL, "Fermez-moi pour terminer...", NULL, NULL,
```

```
    0, 0, 0, 0, WBENCHSCREEN
};

SHORT Palette[] = { 0x0000, 0x0AAA };

struct Interrupt VBLInt;
struct IntData intData;

/*
* Prototypes
*/
extern void VBLServeur(void);  /* Ca, c'est le serveur */
void main(int argc, char **argv);
void cleanExit(UBYTE *str);

/*
* C'est parti !
*/
void main(int argc, char **argv)
{
UBYTE buf[80];
int i;
struct IntuiMessage *im;

    if (!argc)  /* Lancement depuis le CLI seulement */
        exit(20);   /* (because on utilise printf)       */

    if (!(IntuitionBase = (struct IntuitionBase *)
      OpenLibrary("intuition.library", 33L)))
        cleanExit("Pas d'Intuition !");

    if (!(GfxBase = (struct GfxBase *)
      OpenLibrary("graphics.library", 33L)))
        cleanExit("Pas de Graphics !");

    if (!(win = OpenWindow(&nw)))
        cleanExit("Impossible d'ouvrir la fenêtre !");

    if (!(scr = OpenScreen(&ns)))
        cleanExit("Impossible d'ouvrir l'écran !");

    LoadRGB4(&scr->ViewPort, Palette, 1L << DEPTH);

/*
* Remplissage de la structure qui sera transmise au serveur...
*/
    intData.id_Bitplane = scr->BitMap.Planes[0];
    intData.id_BplSize  = scr->BitMap.BytesPerRow * scr->BitMap.Rows;
    intData.id_Current  = intData.id_Bitplane;
    intData.id_BplCount = intData.id_BplSize;

    VBLInt.is_Node.ln_Type = NT_INTERRUPT;
    VBLInt.is_Node.ln_Name = "ANT";
    VBLInt.is_Node.ln_Pri  = 0;    /* Priorité nominale */
    VBLInt.is_Data         = (APTR)&intData;
    VBLInt.is_Code         = VBLServeur;
    AddIntServer(INTB_VERTB, &VBLInt);

    SetAPen(win->RPort, 1);
    SetDrMd(win->RPort, JAM2);
    Move(win->RPort, 10, 20);
    Text(win->RPort,
          "Pendant que le serveur VBL s'amuse dans son écran...",
          52);

    for(i = 0;;)
    {   WaitPort(win->UserPort);
        while (im = (struct IntuiMessage *)GetMsg(win->UserPort))
        {   switch(im->Class)
            {   case CLOSEWINDOW:
                    RemIntServer(INTB_VERTB, &VBLInt);
                    cleanExit(NULL);
                    break;

                case INTUITICKS:
                    sprintf(buf,
                            "Moi, je compte les INTUITICKS : %3d",
                            ++i);
                    Move(win->RPort, 10, 30);
                    Text(win->RPort, buf, 35);
                    break;
            }
            ReplyMsg((struct Message *)im);
        }
    }
}

/*
*
*/
```

```
void cleanExit(UBYTE *str)
{
    if (scr)
        CloseScreen(scr);

    if (win)
        CloseWindow(win);

    if (GfxBase)
        CloseLibrary((struct Library *)GfxBase);

    if (IntuitionBase)
        CloseLibrary((struct Library *)IntuitionBase);

    if (str)
        puts(str);

    exit(0);
}
```

LISTING 2

```
    opt c+,l+,o+,ow-

; ********************************
; * Serveur.s                    *
; * Serveur d'interruption VBL *
; * S. Schreiber - 030590        *
; ********************************

    XDEF    _VBLServeur ; On exporte ce label

; ***** Définition de notre structure IntData
    rsreset
IntData rs.b    0
id_Bitplane rs.l    1
id_Current  rs.l    1
id_BplSize  rs.w    1
id_BplCount rs.w    1
id_SIZEOF   rs.w    0

; ***** Début du serveur
_VBLServeur:
    movea.l id_Current(a1),a0  ; prend l'adresse courante
    eori.l  #-1,(a0)+
    move.w  id_BplCount(a1),d0
    subq.w  #4,d0   ; 1 mot long de moins
    beq.s   Reset
Retour  move.l  a0,id_Current(a1)
    move.w  d0,id_BplCount(a1)
    moveq   #0,d0   ; Et la chaîne continue...
    rts

Reset   move.l  id_Bitplane(a1),id_Current(a1)
    move.w  id_BplSize(a1),id_BplCount(a1)
    moveq   #0,d0
    rts
```

LISTING 3

```
#makefile pour IntServ
EXE=IntServ
OBJ=IntServ.o Serveur.o
LIB=LIB:lc.lib LIB:amiga.lib
ARG=-b1 -cfist -v -y
LNK=SC SD DEFINE __main=__tinymain NOICONS NODEBUG

IntServ:     $(OBJ)
BLink LIB:c.o $(OBJ) TO $(EXE) LIB $(LIB) $(LNK)

IntServ.o:   IntServ.c
Lc $(ARG) IntServ.c
```

LA MEMOIRE OUBLIEE

Philippe Vautrin ayant apparemment oublié d'emporter son Amiga en vacances, la suite de son article sur la gestion de la mémoire par Exec ne nous est pas parvenue à temps pour paraître dans ce numéro. Nous prions donc nos abonnés de nous excuser du désagrément que celà peut leur causer. Ils retrouveront la suite de ce pourtant excellent article dans notre prochain numéro.

La Rédaction.

Par Loïc FAR

SubWAY II : Bougez avec intui-

Suite au dernier article sur la gestion des sprites hardware, nous allons ce mois-ci poursuivre notre exploration du monde de l'animation sur Amiga.

Le mois dernier, nous avons vu qu'il était possible d'animer jusqu'à huit sprites hardware simultanément sur l'écran. Mais que pouvons nous faire si nous désirons en avoir plus ? Et bien il existe une solution, ce sont les VSprites, pour "Virtual Sprite".

LES VSPRITES, QUI SONT-ILS ?

Le VSprite est la plus petite entité logicielle utilisable pour l'animation. Il repose d'ailleurs presque entièrement sur les Sprites hardware. Lorsque vous réservez un Sprite hardware, celui-ci vous appartient entièrement, jusqu'à ce que vous indiquiez explicitement au système que vous le libérez (par la fonction FreeSprite). Avec un VSprite, c'est différent. Le système prend à sa charge la gestion de la partie hardware, c'est-à-dire seulement lorsque vous décidez d'afficher ce VSprite à l'écran. Il n'allouera à ce VSprite un vrai sprite hardware que durant le temps nécessaire à son affichage. Cela signifie que vous pouvez très bien afficher trois VSprites en utilisant le même sprite hardware si ceux-ci sont suffisamment bien disposés.

Du point de vue programmation, un VSprite est représenté par une structure C qui définit sa taille, son "look", ses couleurs, ses coordonnées à l'écran et la conduite à tenir en cas collsion.

Pour créer un VSprite (cf. programme ci-joint) les étapes à réaliser sont donc les suivantes :

1) Initialiser le système GEL (Graphic ELements) une bonne fois pour toute avec la fonction InitGel() ;

2) Définir le VSprite (hauteur, position à l'écran, image, couleurs, flags) ;

3) Ajouter le VSprite à la liste des GELs.

Il est alors possible de changer sa forme, sa couleur ou sa position en modifiant les paramètres adéquats associés à l'élément graphique. Je vous conseille d'étudier le programme ci-dessous qui reprend un à un les points cités ci-dessus.

CONCLUSION

La partie théorique de cet article est extrèmement courte, le programme d'exemple prenant cette fois-ci une bonne partie de la place qui m'est réservée. Mais rassurez-vous, il se continuera le mois prochain par une partie consacrée à la gestion des collisions où j'en profiterai pour détailler un peu plus la gestion de nos petits VSprites.

Bonne Animation et à bientôt.

```
/* ------------------------------------------------------------ */
/*  Programme de gestion de Vsprites sous Workbench             */
/*                                                              */
/*  Pprogrammation des VSprites sous Intuition & Graphics       */
/*          - VSprites                                          */
/*          - Detection de collisions (initialisation)          */
/*          - Gestion de la souris                              */
/*          - Création de structures Image                      */
/*                                        P. AMIABLE 1991       */
/* ------------------------------------------------------------ */

/* ------------------------------------------------------------*/
/*  Quelques includes utiles.....                               */
/* ------------------------------------------------------------*/
#include <exec/types.h>
#include <exec/memory.h>
#include <intuition/intuition.h>
#include <stdio.h>
#include <graphics/gels.h>
#include <graphics/gfxmacros.h>

/* ------------------------------------------------------------*/
/*  Les defines maintenant...                                   */
/* ------------------------------------------------------------*/
#define INTUITION_REV 0L /* Version d'Intuition (la dernière)    */
#define GFX_REV       0L /* Version de Graphics (la dernière)    */
#define NBRSPRITES    10 /* Nb VSprites affichés simultanément   */

/* ------------------------------------------------------------*/
/*  Définition des variables externes                           */
/* ------------------------------------------------------------*/

/* Declararation des fonctions définies dans le programme */
void main(), referme(), init();
void detruit_vsprites();
void ouvrefenetre();
void refermegel();
int initgel();
void graphique();
void deplace();
int bidon();

struct IntuitionBase *IntuitionBase = NULL;
struct GfxBase *GfxBase = NULL;
struct Window *fenetre = NULL;
struct Screen *ecran = NULL;
struct RastPort *rastport = NULL;
struct ViewPort *viewport  = NULL;
struct VSprite *tete = NULL;
struct VSprite *queue = NULL;

struct NewScreen nouvecran = {
    0, 0,          /* LeftEdge, TopEdge                        */
    320, 256,      /* Width, Height                            */
    5,             /*  Depth, soit 32 couleurs possibles       */
    1, 0,          /* DetailPen, BlockPen                      */
    SPRITES,       /* ViewModes valeur 0 = basse resolution    */
    CUSTOMSCREEN,  /* Type d'ecran                             */
    NULL,          /* Font -> NULL = fonte par defaut          */
    "titre ",      /* DefaultTitle pour le titre -> aucun      */
    NULL,          /* gadgets, aucun                           */
    NULL };        /* adresse de la CustomBitmap -> aucune ici*/

struct NewWindow nouvfenetre =
{
    0, 0,               /* LeftEdge, TopEdge                   */
    150, 15,            /* Width, Height                       */
    0, 1,               /* DetailPen, BlockPen                 */
    NULL,               /* IDCMPFlags                          */
    SMART_REFRESH|      /* Flags                               */
        ACTIVATE,
    NULL,               /* FirstGadget                         */
    NULL,               /* CheckMark                           */
    "Virtual Sprites",  /* Title                               */
    NULL,               /* Screen                              */
    NULL,               /* BitMap                              */
    150,                /* MinWidth                            */
    15,                 /* MinHeight                           */
    150,                /* MaxWidth                            */
    15,                 /* MaxHeight                           */
    CUSTOMSCREEN        /* Type                                */
};

SHORT vitesse[]={ 1, 2, -1, -2 }; /* Echelle de vitesse utilisée */
```

-tion et Graphics (II)

```
SHORT xmouv[NBRSPRITES], ymouv[NBRSPRITES];     /* Directions     */
short max_cree = 0;                /* nombre maxi de sprites crées */
struct VSprite *vsprite[NBRSPRITES];
struct GelsInfo gelinfo;

/* Données du sprite */
UWORD chip sprite[]=
{
    0x0000,0x3000,0x0000,0x7800,0x0000,0xcc00,0x0000,0xcc00,
    0x0000,0xcc00,0x0460,0xfc60,0x067e,0xfe00,0x077e,0xcf00,
    0x07f8,0xcfe0,0x06f8,0xcee0,0x0678,0xce60,0x0678,0x0660,
    0x0678,0x0660,0x0018,0x0000
};

UWORD tablecouleur[] = {
    0x0000,0x0e30,0x0fff,0x0b40,0x0fb0,0x0bf0,0x05d0,0x0ed0,
    0x07df,0x069f,0x0c0e,0x0f2e,0x0feb,0x0c98,0x0bbb,0x07df,
    0x0000,0x0e30,0x0fff,0x0b40,0x0fb0,0x0bf0,0x05d0,0x0ed0,
    0x07df,0x069f,0x0c0e,0x0f2e,0x0feb,0x0c98,0x0bbb,0x07df
};

UWORD palette0[] = { 0x0e30, 0xffff, 0x0b40 };
UWORD palette1[] = { 0x0bf0, 0x05d0, 0x0ed0 };
UWORD palette2[] = { 0x069f, 0x0c0e, 0x0f2e };
UWORD palette3[] = { 0x0c98, 0x0bbb, 0x07df };
UWORD *palette[] = { palette0, palette1, palette2, palette3 };
int choix_couleur;

/* Port A du CIA dont le bit - bouton gauche de la souris */
char *bouton_gauche = (char *) 0xbfe001;

/* ------------------------------------------------------------------*/
/*         Programme principal                                       */
/* ------------------------------------------------------------------*/
void main()
{
int i = 0;

    init();              /* Ouverture des bibliothèques système   */
    ouvrefenetre();      /* Ouverture de l'écran et de la fenêtre */
    initgel();           /* initialisation du système GEL         */
    graphique();

    while(!((*bouton_gauche & 0x40)-64))
    {   deplace();       /* On bouge les sprites                  */
        WaitTOF();       /* Attente de trame                      */
    }

    for(i = 0; i < max_cree; i++)
        detruit_vsprites(vsprite[i]); /* Détruit les VSprites     */

    refermegel();            /* on désalloue la structure gelinfo */
    referme();               /* on referme le reste               */
}

/* ------------------------------------------------------------------*/
/*  Init() Ouvre la bibliothèque                                     */
/* ------------------------------------------------------------------*/
void init()
{
char *OpenLibrary();

    IntuitionBase = (struct IntuitionBase *)
      OpenLibrary("intuition.library",INTUITION_REV);
    if(!IntuitionBase)
        exit(FALSE);

    GfxBase = (struct GfxBase *)
      OpenLibrary("graphics.library",GFX_REV);
    if(!GfxBase)
    {   referme();
        exit(FALSE);
    }
}

/* ------------------------------------------------------------------*/
/*  ouvrefenetre() ouvre la fenetre et l'ecran                       */
/* ------------------------------------------------------------------*/

void ouvrefenetre()
{
void referme();

    if(!(ecran = (struct Screen*)OpenScreen(&nouvecran)))
    {   referme();
        exit(FALSE);
    }
    nouvfenetre.Screen = ecran;

    if(!(fenetre=(struct Window*)OpenWindow(&nouvfenetre)))
    {   referme();
        exit(FALSE);
    }

    rastport = fenetre->RPort;
    viewport = (struct ViewPort *)ViewPortAddress(fenetre);
    LoadRGB4(viewport, &tablecouleur[0], 32);
}

/* ------------------------------------------------------------------*/
/*  Cette fonction referme la fenetre et les bibliothèques           */
/* ------------------------------------------------------------------*/
void referme()
{
    if(fenetre)       CloseWindow(fenetre);
    if(ecran)         CloseScreen(ecran);
    if(IntuitionBase) CloseLibrary(IntuitionBase);
    if(GfxBase)       CloseLibrary(IntuitionBase);
}

/* ------------------------------------------------------------------*/
/*  Cette fonction déplace les max_cree sprites                      */
/* ------------------------------------------------------------------*/
void deplace()
{
short i;

    for (i=0; i<max_cree; i++)
    {   vsprite[i]->X = xmouv[i]+vsprite[i]->X;
        vsprite[i]->Y = ymouv[i]+vsprite[i]->Y;

        /* Test de dépassement de l'écran */
        if(vsprite[i]->X >= 290 || vsprite[i]->X <= 0)
            xmouv[i]=-xmouv[i];
        if(vsprite[i]->Y >= 230 || vsprite[i]->Y <= 0)
            ymouv[i]=-ymouv[i];
    }

    SortGList(rastport);   /* On réordonne la liste des VSprites */
    DrawGList(rastport, viewport);

    MakeScreen(ecran);  /* on régénère l'écran                   */
    RethinkDisplay();   /* et on visualise le tout               */
}

/* ------------------------------------------------------------------*/
/*  Routine appelée lors de la détection de collision                */
/* ------------------------------------------------------------------*/
/* on étudiera cela plus en détail le mois prochain */
int bidon()
{
    return(0); /* pour l'instant, on ne fait rien...              */
}

/* ------------------------------------------------------------------*/
/*  Cette fonction initialise la structure Gels.                     */
/* ------------------------------------------------------------------*/
int initgel()
{
    if ((tete = (struct VSprite *)AllocMem(sizeof(struct VSprite),
      MEMF_PUBLIC | MEMF_CLEAR)) == 0)
        return(-1);

    if ((queue = (struct VSprite *)AllocMem(sizeof(struct VSprite),
      MEMF_PUBLIC | MEMF_CLEAR)) == 0)
    {   refermegel();
        FreeMem(tete, sizeof(struct VSprite));
        return(-2);
    }

    /* Les sprites 0 et 1 sont réservés pour Intuition           */
    gelinfo.sprRsrvd = 0xFC;

    if ((gelinfo.nextLine = (WORD *)AllocMem(sizeof(WORD) * 8,
      MEMF_PUBLIC | MEMF_CLEAR)) == NULL)
    {   refermegel();
        FreeMem(tete, sizeof(struct VSprite));
        FreeMem(queue, sizeof(struct VSprite));
        return(-3);
    }
```

```
if ((gelinfo.lastColor = (WORD **)AllocMem(sizeof(LONG) * 8,
  MEMF_PUBLIC | MEMF_CLEAR)) == NULL)
{   refermegel();
    FreeMem(tete, sizeof(struct VSprite));
    FreeMem(queue, sizeof(struct VSprite));
    return(-4);
}

/* On prépare une table de pointeurs vers des routines qui
 * seront exécutées quand la fonction DoCollision détectera
 * une collision.
 * Nous étudierons le fonctionnement d'un tel système en
 * détails le mois prochain
 */
if (!(gelinfo.collHandler = (struct collTable *)
  AllocMem(sizeof(struct collTable), MEMF_PUBLIC|MEMF_CLEAR)))
{   refermegel();
    FreeMem(tete, sizeof(struct VSprite));
    FreeMem(queue, sizeof(struct VSprite));
    return(-5);
}

/* quand une partie d'un objet passe à travers les frontières,
 * il déclenche l'appel à la routine de gestion de collision
 * (pour le moment, bidon())
 */
    gelinfo.leftmost = 0;
    gelinfo.rightmost = rastport->BitMap->BytesPerRow * 8 - 1;
    gelinfo.topmost = 0;
    gelinfo.bottommost = rastport->BitMap->Rows - 1;

    rastport->GelsInfo = &gelinfo;
    InitGels(tete, queue, &gelinfo);
    SetCollision(0, bidon, &gelinfo);
    WaitTOF();
}

/* ------------------------------------------------------------- */
/*  Cette fonction désalloue la structure gels .                 */
/* ------------------------------------------------------------- */
void refermegel()
{
    if (gelinfo.collHandler)
        FreeMem(gelinfo.collHandler, sizeof(struct collTable));
    if (gelinfo.lastColor)
        FreeMem(gelinfo.lastColor, sizeof(LONG) * 8);
    if (gelinfo.nextLine)
        FreeMem(gelinfo.nextLine, sizeof(WORD) * 8);
    if (gelinfo.gelHead)
        FreeMem(gelinfo.gelHead, sizeof(struct VSprite));
    if (gelinfo.gelTail)
        FreeMem(gelinfo.gelTail, sizeof(struct VSprite));
}

/* ------------------------------------------------------------*/
/*  Cette fonction crée un VSprite.                            */
/* ------------------------------------------------------------*/
struct VSprite *cree_vsprites(hauteur, image, tablecol, x, y)
SHORT hauteur;  /* hauteur du VSprite en nombre de lignes      */
WORD *image;    /* dessin du VSprite (c'est une table de mots) */
WORD *tablecol; /* couleurs du VSPrite                         */
SHORT x, y;     /* position intiale du sprite                  */
{
struct VSprite *vsprite;

    if ((vsprite=(struct VSprite *)AllocMem(sizeof(struct VSprite),
      MEMF_PUBLIC | MEMF_CLEAR)) == NULL)
        return(0);

    vsprite->Flags  = VSPRITE; /* C'est un VSprite et non un Bob */
    vsprite->Y      = y;
    vsprite->X      = x;
    vsprite->Height = hauteur;
    vsprite->Width  = 1;  /* Toujours 16 pixels (1 mot) de large */
    vsprite->Depth  = 2;  /* Toujours quatre couleurs            */
    vsprite->MeMask = 1;  /* sert à la détection de collision    */
    vsprite->HitMask = 1;
    vsprite->ImageData = image;    /* on associe l'image au vsprite */

    if ((vsprite->BorderLine = (WORD *)AllocMem(sizeof(WORD),
      MEMF_PUBLIC | MEMF_CLEAR)) == 0)
    {   FreeMem(vsprite, sizeof(struct VSprite));
        return(0);
    }

    if ((vsprite->CollMask = (WORD *)AllocMem(sizeof(WORD)*hauteur,
      MEMF_CHIP | MEMF_CLEAR)) == 0)
    {   FreeMem(vsprite, sizeof(struct VSprite));
        FreeMem(vsprite->BorderLine, sizeof(WORD));
        return(0);
    }

    vsprite->SprColors  = tablecol;
    vsprite->PlanePick  = 0x00;
    vsprite->PlaneOnOff = 0x00;

    InitMasks(vsprite);
    return(vsprite);
}

/* ------------------------------------------------------------*/
/*  Cette fonction détruit un VSprite et libére ses ressources  */
/* ------------------------------------------------------------*/
void detruit_vsprites(vsprite)
struct VSprite *vsprite;
{
register long VMEM;
register APTR ptr;

    if (vsprite != NULL)
    {   VMEM = vsprite->Width * vsprite->Height * vsprite->Depth;
        if (vsprite->VSBob != NULL)
        {   if (vsprite->VSBob->SaveBuffer != NULL)
                FreeMem(vsprite->VSBob->SaveBuffer,   sizeof(SHORT)   *
VMEM);
            if (vsprite->VSBob->DBuffer != NULL)
            {   if (vsprite->VSBob->DBuffer->BufBuffer != 0)
                    FreeMem(vsprite->VSBob->DBuffer->BufBuffer,
sizeof(SHORT) * VMEM);
                FreeMem(vsprite->VSBob->DBuffer,          sizeof(struct
DBufPacket));
            }
            FreeMem( vsprite->VSBob, sizeof(struct Bob));
        }
        if (vsprite->CollMask != NULL)
            FreeMem(vsprite->CollMask,
sizeof(WORD)*vsprite->Height*vsprite->Width);
        if (vsprite->BorderLine != NULL)
            FreeMem(vsprite->BorderLine, sizeof(WORD)*vsprite->Width);
        FreeMem(vsprite, sizeof(struct VSprite));
    }
}

/* ------------------------------------------------------------*/
/*  Cette fonction crée l'ensemble des éléments graphiques      */
/* ------------------------------------------------------------*/
void graphique()
{
int erreur, i, x, y;

    erreur = initgel(&gelinfo, rastport);

    for(i=0; i < NBRSPRITES; i++)
    {   x = 10 + RangeRand(280);
        y = 10 + RangeRand(230);

        xmouv[i] = vitesse[RangeRand(4)];
        ymouv[i] = vitesse[RangeRand(4)];

        vsprite[i] = cree_vsprites(14,sprite,
                                   palette[choix_couleur], x, y);
        if(vsprite[i] == 0)
            break;

        AddVSprite(vsprite[i], rastport);

        max_cree++;
        choix_couleur++;
        if(choix_couleur >= 4)
            choix_couleur = 0;
    }
}
```

par Herr DOKTOR

TRANSACTOR : LES ROULEAUX

La routine que nous vous proposons aujourd'hui dans Transactor nous a été envoyée par François Brion, de la banlieue Dijonnaise. Il s'agit de rouleaux de couleurs animés, à ne pas confondre avec les Rasters, dont ils sont pourtant dérivés.

Avant que de laisser la parole à François, rappellons simplement le pourquoi de cette rubrique Transactor, qui donc remplace le défunt Concours Lecteurs : Transactor vous donne la possibilité de vous exprimer librement sur tout sujet que vous possédez pleinement, tout comme le font nos journalistes réguliers. Il peut s'agir d'expliquer une routine particulière, comme François a choisi de le faire ici, ou une technique de programmation, l'utilisation de tel ou tel élément hardware ou logiciel de l'Amiga, etc. Encore une fois, tous les sujets sont possibles.

L'écriture d'un article pour Transactor est rémunérée au tarif forfaitaire de 1,500 F HT les deux pages. Vous trouverez un bon de participation, ainsi que le régèlement général, en page 32. Mais je sens que François commence à s'impatienter, il est grand temps de lui passer la plume.

LE COPPER, CA ROULE !

La routine que je vous propose permet de réaliser très simplement un effet impressionant : des rasters multicolores qui donnent l'impression de s'enrouler sur eux-même d'où le terme "rouleau" que j'utilise.

Le principe de la routine est très simple mais l'effet est très joli.

En effet, il suffit mettre à la suite dans la CopperList une série d'instructions MOVE avec le registre de couleur voulu (on prendra en général COLOR00) et le dégradé recherché (j'ai utilisé dans le programme un dégradé bleu et rouge). Il suffira ensuite de décaler toutes les couleurs dans le CopperList pour donner l'impression que les barres s'enroulent.

Le Copper a besoin de deux cycles pour exécuter une instruction MOVE. Comme le DMA bitplan affiche 2 pixels LowRes par cycle, la couleur ne sera modifiée que tous les 4 pixels. De plus, le Copper ne peut exécuter que 56 instructions MOVE par ligne, mais comme le dégradé a besoin de 60 couleurs pour être complet, il faut le répéter 13 fois sinon la dernière ligne s'arrêtera en plein milieu de l'écran (le rouleau complet fait alors 14 lignes de raster de haut).

Pour réaliser l'animation des couleurs du rouleau, le 68000 ou le blitter peuvent être utilisés au choix. Dans ce programme, j'ai choisi le blitter. Il faut sauvegarder dans un registre la valeur de la première couleur, réaliser un décalage de toute la partie de la CopperList qui concerne le rouleau, et remettre la couleur sauvegardée à la fin.


```
;
; François BRION pour ANT
;
   optc+,o+,ow-

AllocMem=-198
FreeMem=-210
FindTask=-294
CloseLibrary=-414
OpenLibrary=-552

SysCop1=$26
SysCop2=$32

Custom=$DFF000
DMACONR=$002
VPOSR=$004
VHPOSR=$006
POTGOR=$016
INTENAR=$01C
INTREQR=$01E
COP1LCH=$080
COP1LCL=$082
COP2LCH=$084
COP2LCL=$086
COPJMP1=$088
COPJMP2=$08A
DIWSTRT=$08E
DIWSTOP=$090
DDFSTRT=$092
DDFSTOP=$094
DMACON=$096
INTENA=$09A
INTREQ=$09C
BPL1PTH=$0E0
BPL1PTL=$0E2
BPL2PTH=$0E4
BPL2PTL=$0E6
BPL3PTH=$0E8
BPL3PTL=$0EA
BPL4PTH=$0EC
BPL4PTL=$0EE
BPL5PTH=$0F0
BPL5PTL=$0F2
BPL6PTH=$0F4
BPL6PTL=$0F6
BPLCON0=$100
BPLCON1=$102
BPLCON2=$104
BPL1MOD=$108
BPL2MOD=$10A
SPR0PTH=$120
SPR0PTL=$122
SPR1PTH=$124
SPR1PTL=$126
SPR2PTH=$128
SPR2PTL=$12A
SPR3PTH=$12C
SPR3PTL=$12E
SPR4PTH=$130
SPR4PTL=$132
SPR5PRH=$134
SPR5PTL=$136
SPR6PTH=$138
SPR6PTL=$13A
SPT7PTH=$13C
SPR7PTL$=13E
COLOR00=$180
COLOR01=$182
COLOR02=$184
COLOR03=$186
COLOR04=$188
COLOR05=$18A
COLOR06=$18C
COLOR07=$18E
COLOR08=$190
COLOR09=$192
COLOR10=$194
COLOR11=$196
COLOR12=$198
COLOR13=$19A
COLOR14=$19C
COLOR15=$19E
COLOR16=$1A0
COLOR17=$1A2
COLOR18=$1A4
COLOR19=$1A6
COLOR20=$1A8
```

```
COLOR21=$1AA
COLOR22=$1AC
COLOR23=$1AE
COLOR24=$1B0
COLOR25=$1B2
COLOR26=$1B4
COLOR27=$1B6
COLOR28=$1B8
COLOR29=$1BA
COLOR30=$1BC
COLOR31=$1BE

CIAAPRA=$BFE001

DEGRADE=60  ; nb couleurs dans le dégradé
NB_DEGR=13  ; nb répétitions du dégradé dans 1 rouleau
ROLSIZE=DEGRADE*NB_DEGR*2  ; taille du rouleau en octets

; ************************
; * DEBUT DU PROGRAMME *
; ************************
Start   movea.l $4.w,a6

    lea gfxname,a1
    moveq   #0,d0
    jsr OpenLibrary(a6)
    move.l  d0,a1
    move.l  SysCop1(a1),oldcop1
    move.l  SysCop2(a1),oldcop2
    jsr CloseLibrary(a6)

    lea $dff000,a6

    move.w  DMACONR(a6),d0
    ori.w   #$8200,d0
    move.w  d0,olddma

    move.w  INTENAR(a6),d0
    ori.w   #$c000,d0
    move.w  d0,oldint

    move.w  #$7fff,DMACON(a6)
    move.w  #$7fff,INTENA(a6)
    move.w  #$7fff,INTREQ(a6)

    lea Roller1,a0  ; crée le 1er rouleau
    bsr MakeRoller
    lea Roller2,a0  ; crée le 2eme rouleau
    bsr MakeRoller

    lea NewCop,a0
    move.l  a0,COP1LCH(a6)
    move.w  a0,COPJMP1(a6)

    move.w  #$8280,DMACON(a6)  ; dma copper seulement

; ************************
; * BOUCLE PRINCIPALE *
; ************************
Vloop   btst    #5,INTREQR+1(a6)    ; attend le vbl
    beq.s   Vloop
    move.w  #$20,INTREQ(a6)

    bsr MoveRollers

    btst    #6,CIAAPRA
    bne.s   Vloop

Ciao    move.w  #$7fff,DMACON(a6)
    move.w  #$7fff,INTENA(a6)
    move.w  #$7fff,INTREQ(a6)
    move.l  oldcop1(pc),COP1LCH(a6)
    move.l  oldcop2(pc),COP2LCH(a6)
    move.w  d0,COPJMP1(a6)
    move.w  olddma(pc),DMACON(a6)
    move.w  oldint(pc),INTENA(a6)

    moveq   #0,d0
    rts

; ************************************
; * CREE LE MOUVEMENT DES ROULEAUX *
; ************************************
MoveRollers:
    lea Roller1+(ROLSIZE*2),a0 ; 1er rouleau
    lea -4(a0),a1
    move.w  -2(a0),d0
    move.w  #(ROLSIZE/2)-2,d1
Avant   move.l  -(a1),-(a0)
    dbra    d1,Avant
    move.w  d0,2(a0)

    lea Roller2,a0      ; 2eme rouleau
    lea 4(a0),a1
    move.w  2(a0),d0
    move.w  #(ROLSIZE/2)-2,d1
Arriere move.l  (a1)+,(a0)+
    dbra    d1,Arriere
    move.w  d0,-2(a1)
    rts

; ****************************************
; * CREE LES ROULEAUX DANS LA COPPERLIST *
; ****************************************
MakeRoller:
    moveq   #NB_DEGR-1,d0
.make   moveq   #0,d1

    moveq   #14,d2
.roll1  move.w  #$0180,(a0)+    ; rouge 0 -> rouge 15
    move.w  d1,(a0)+
    addi.w  #$0100,d1
    dbra    d2,.roll1
    move.w  #$0180,(a0)+
    move.w  d1,(a0)+

    moveq   #14,d2
.roll2  move.w  #$0180,(a0)+    ; rouge 14 -> rouge 0
    subi.w  #$0100,d1
    move.w  d1,(a0)+
    dbra    d2,.roll2

    moveq   #14,d2
.roll3  move.w  #$0180,(a0)+    ; bleu 1 -> bleu 15
    addq.w  #$0001,d1
    move.w  d1,(a0)+
    dbra    d2,.roll3

    moveq   #13,d2
.roll4  move.w  #$0180,(a0)+    ; bleu 14 -> bleu 0
    subq.w  #$0001,d1
    move.w  d1,(a0)+
    dbra    d2,.roll4

    dbra    d0,.make
    rts

; ************************
; * DONNES EN FAST-RAM *
; ************************
gfxname dc.b    "graphics.library",0
    even

oldcop1 ds.l    1
oldcop2 ds.l    1
olddma  ds.w    1
oldint  ds.w    1

; ************************
; * DONNEES EN CHIP-RAM *
; ************************
    section CHIP,DATA_C

NewCop  dc.w    DIWSTRT,$2c81,DIWSTOP,$2cc1
    dc.w    DDFSTRT,$0038,DDFSTOP,$00d0
    dc.w    BPLCON0,$0000
    dc.w    COLOR00,$0000

    dc.w    $2c31,$fffe
Roller1 ds.w    ROLSIZE
    dc.w    COLOR00,$0000

    dc.w    $d931,$fffe
Roller2 ds.w    ROLSIZE
    dc.w    COLOR00,$0000

    dc.l    -2

    END
```

par F.BRION

REQUESTER

Après avoir laissé large place à ceux qui désiraient exprimer leur avis sur l'avenir du magazine, le Requester reprend sa vocation initiale, à savoir répondre à vos questions de programmation.

Salut à tous ! L'objectif de cette lettre ne consiste pas seulement à vous poser des questions, mais dans un premier temps, j'expose mon avis sur le journal et réponds à vos appels.
Si je me suis récemment ré-abonné, c'est que grâce à l'ANT, je réussis de mieux en mieux l'apprentissage de l'assembleur. Toutefois, il faut bien voir que c'est le seul journal de programmation en français et par conséquent, il est normal que l'on attende beaucoup de lui ! Notamment - et vous le savez - son prix relativement élevé, qui a failli me faire obstacle. Sinon, pour un tel prix, l'ANT ne répond pas toujours aux exigences des lecteurs (en tout cas des miennes) et l'on se sent parfois un peu frustré. Aussi, il est nécessaire qu'il y ait notre contribution dans le journal, et vous l'avez toujours demandée. Mais il faut qu'elle ait une suite concrète. D'autre part, pour faire face à certaines exigences, il faut impérativement que le journal s'agrandisse.
(...)
Je me permets de faire une autre suggestion qui me tiens à coeur : la programmation de jeux m'attire pour son côté complexe, technique et esthétique. C'est pour celà que j'apprécie des rubriques telles que "DemoMakers" et le "Concours" (NDLR : ex-Concours...). Pourtant, je verrais bien une nouvelle rubrique, plus axée sur cette programmation de jeux. Attention, il ne s'agirait pas qu'à chaque numéro, on ait un jeu à taper, mais d'explications, d'aides, d'astuces techniques, par exemple sur la gestion des Bobs, etc. Qu'en pensent vos autres lecteurs ?
(...)
Je suis bloqué pour programmer, par manque de connaissances, et c'est là le deuxième objectif de cette lettre.
- j'aimerais savoir comment on peut apprendre la programmation de la CopperList, dont je ne connais absolument rien ;
- par contre, je sais programmer le Blitter, mais lors de transferts, mon plus gros soucis, c'est l'adresse de destination pour un écran ouvert sous Intuition. J'ai beau savoir "qu'on récupère depuis la structure de l'écran, l'adresse du bitplan" (cf. CR 31), où ceci se trouve-t-il précisément, à l'octet près ?
- le blitter peut-il s'utiliser dans une fenêtre ouvertre sous Intuition et si oui, là encore, quelle est l'adresse exacte de destination ?
- puisqu'on est dans la programmation du Blitter, restons-y ! L'affichage de Bobs nécessite un transfert vers chaque bitplan ; or, le démarrage du Blitter représente une importante part du temps machine, alors ne faire qu'un seul transfert apporte un gain de temps essentiel. La méthode permettant ceci consiste à mettre les unes à la suite des autres les lignes du bitplan 1, du bitplan 2, etc. L'affichage de fait alors en indiquant un modulo de fin de ligne. J'aimerais que vous expliquiez cette méthode en donnant comme exemple le transfert d'un Bob. Pourquoi d'ailleurs ne pas en faire un rubrique à part ?
- pour bien des programmes proposés dans l'ANT, les Bobs proviennent d'une image au format Raw, que l'on peut obtenir par Deluxe IFFConverter, fourni avec DeluxePaint. Le problème est que je ne possède pas un tel utilitaire, et que je ne vais pas acheter un autre utilitaire de dessin juste pour ça. Il serait donc intéressant de publier un programme de ce type, ce qui permettrait d'en apprendre le fonctionnement.
- enfin, j'aimerais connaître la gestion complète des joysticks pour les deux ports, sans pour autant passer par les bibliothèques.
J'aurais beaucoup d'autres questions à vous poser (par exemple, nombre de fonctions de la graphics.library me sont encore obscures) mais il fallait bien sélectionner ! Oh, j'allais oublier une toute dernière : le fait d'avoir ouvert un écran sous Intuition ralentit-il l'exécution d'un programme par rapport à un écran créé avec le Copper ?

Nicolas Liquière, Le Truel

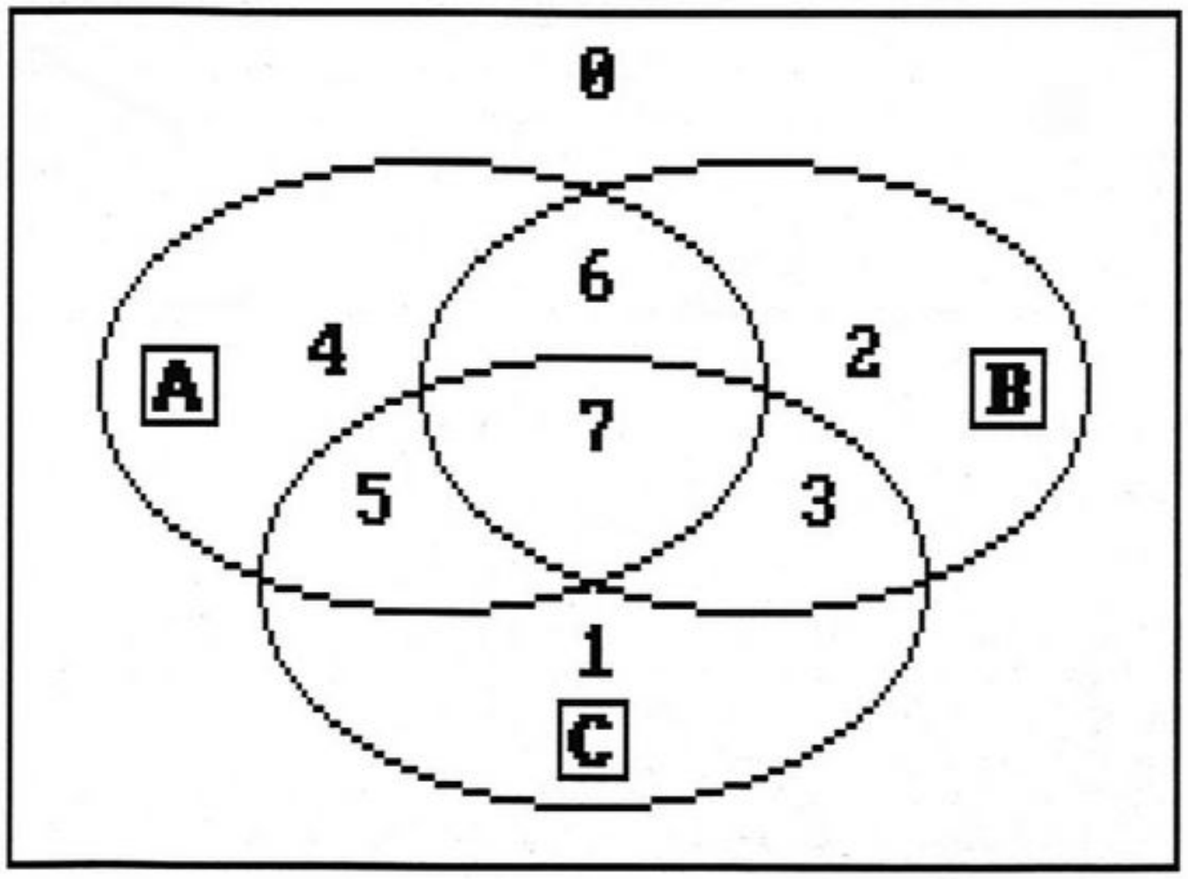
Fichtre, diantre, que de questions dans cette très longue lettre, dont nous avons pourtant coupé quelques passages relatifs à l'ex-proposition de changement de formule d'abonnement et aux RKM en français. Prenons donc le taureau par les cornes et notre courage à deux mains et essayons de répondre le plus précisément possible.
Une rubrique sur la programmation des jeux est une excellente idée, et nous allons y réfléchir. Le seul problème étant alors de trouver quelqu'un de suffisamment pointu dans ce domaine pour tenir cette rubrique. Si un programmeur d'une maison d'édition quelconque (nous ne sommes pas sectaires) lit ce courrier, qu'il nous contacte !
La programmation du Copper s'apprend... en lisant la documentation adéquate ! Je ne peux que vous conseiller, pour commencer, la Bible de l'Amiga, aux éditions Micro-Application ; vous y trouverez un chapître complet sur le Copper qui pour une fois, n'est pas truffé d'erreurs.
L'adresse des bitplans d'un écran Intuition se trouve effectivement dans la structure Screen renvoyée par la fonction OpenScreen(). Le premier bitplan se trouve à l'octet $C0, le second à $C4, le troisième à $C8, etc.
Le Blitter peut effectivement s'utiliser directement dans une fenêtre Intuition, mais ce n'est là une méthode ni pratique, ni très sure. Si vous y tenez absolument, le mieux est encore d'ouvrir une fenêtre de type SUPER_BITMAP, pour laquelle vous précisez vous-même l'adresse des bitplans à utiliser, bitplans que vous aurez pu allouer aupravant avec AllocRaster(), par exemple.
La technique des "super plans" n'est plus un secret pour personne... Elle sera expliquée en détails dans notre prochain numéro. Quant à en faire une rubrique à part, c'est déjà fait ! Le Transactor, digne successeur du Concours Lecteurs, est là pour ça !
Un IFF Converter n'est pas très difficile à programmer en soi ; le plus hardu dans l'histoire étant de trouver les routine de chargement et d'affichage d'une image IFF. Une telle routine à déjà été proposée dans l'ANT, alors qu'il faisait encore partie de Commodore Revue. Sinon, le ScreenSaver de Max, proposé dans le cadre de la rubrique Utilitaire du numéro 24, peut être une excellente base à un tel programme : au lieu de sauvegarder l'image en IFF, sauvez-là simplement au format "brut" !
Quant à la gestion des joysticks, elle a, justement, été décrite dans notre dernier numéro, rubrique Transactor.
Enfin, il est évident qu'utiliser le système d'exploitation ralentit quelque peu l'exécution des programmes. D'abord parce ledit système est multitâche (votre programme ne dispose donc pas de tout le temps machine pour lui tout seul) et ensuite, parce qu'il est obligé de gérer pas mal de trucs dont on n'a certainement pas besoin dans un jeu.

Bonjour à tous ! C'est du fin fond de ma cambrousse que je vous écris... Où j'habite, il est difficile de se procurer de la documentation sur l'Amiga, exception faite des revues chez les libraires. En cherchant un peu, j'ai tout de même réussi à trouver La Bible de l'Amiga (ancienne édition), mais à part ça... Je n'ose même pas espérer trouver un jour ces fameux Rom Kernal Manuals, dont vous n'arrêtez de faire l'éloge.
Si je vous dis tout ça, c'est pour vous que vous compreniez bien que je suis réellement tout seul dans mon coin ; je connais bien quelques types au lycée qui ont des Amigas, mais à part les jeux, rien ne les intéresse. Moi, j'ai envie d'aller plus loin avec cet ordinateur, de savoir comment il fonctionne et comment l'utiliser au mieux. Celà passe bien entendu par la programmation.
Après avoir été découragé par l'AmigaBasic, je me suis essayé au GFA, lequel m'a donné entière satisfaction jusqu'à ce que je décide d'aller plus loin encore, et de passer à l'assembleur. Je me serais bien essayé au C (je me suis procuré le Sozobon-C du Domaine Public), mais mon penchant pour les démos m'a naturellement aguillé sur l'assembleur. Ce qui ne m'empêche pas d'utiliser les bibliothèques et le multitâche de temps en temps...
En suivant avec attention vos articles et ceux de vos confrères, je suis arrivé à un niveau correct, mais beaucoup de points me sont encore obscurs. Voici donc quelques questions, telles qu'elles me passent par la

tête, auxquelles j'espère que vous pourrez apporter les réponses.
- comme Frédéric Mazué le faisait justement remarquer dans sa série d'articles "le Blitter de B à R" il y a fort longtemps, la routine de tracé de lignes au Blitter présentée dans La Bible est buggée. Je l'ai corrigée, mais je voudrais maintenant pouvoir effectuer du clipping de droite dans une fenêtre de l'écran. Auriez-vous un algorythme efficace pour ça ?
- l'instruction WAIT du Copper est pratique pour attendre une ligne de balayage particulière, mais je n'ai rien compris à la manière de l'utiliser pour attendre une coordonnée X différente de 0, par exemple, le plein milieu d'une ligne.
- toujours pour le Copper, quelle est l'utilité exacte de l'instruction SKIP, et comment l'utilise-t-on ?
- passons aux sprites hardware. L'utilisation du même sprite sur plusieurs lignes de rasters ne me pose plus aucun problème, mais j'avoue avoir du mal à mettre en oeuvre un sprite 16 couleurs.
- dans quelle mesure le Blitter est-il réellement plus rapide que le 68000 pour les transferts de blocs ? Faut-il l'utiliser en toutes circonstances, ou seulement dans certaines conditions particulières ?
- quelle est la séquence exacte (et la plus efficace) pour attendre le VBLANK, autrement dit le retour de balayage écran ? J'utilise une routine qui lit la valeur de VHPOSR, mais elle n'est pas toujours très précise.
- une petite dernière pour la forme. Pourriez-vous publier un tableau de toutes les combinaisons de MinTerms pour le Blitter, avec leur(s) effet(s) et leur utilisation ?
Voilà. J'espère ne pas vous avoir trop ennuyé avec toutes ces questions et que vous pourrez trouver le temps d'y répondre.

Gréogry Laville, Carbonne

Décidément, c'est la mode des lettres fleuve et des questions en vrac... C'est sans doute l'été qui veut ça.
Il existe plusieurs algorythmes de clipping. Jerôme Etienne vous en a récemment démontré un dans sa rubrique DémoMakers (ANT 22), méthode à laquelle Thomas Landspurg avait apporté sa contribution. Mais d'après l'inénarrable Max, l'algorythme le plus performant est celui que messieurs Cohen et Sutherland ont mis au point. Il est d'ailleurs en train d'en finaliser une mise en pratique, qu'il vous proposera très bientôt dans l'ANT, sans doute dès le prochain numéro.
Le Copper est en effet très pratique pour des effets spéciaux tels qu'un dégradé de couleurs, des barres de couleurs, etc. Malheureusement, un cours complet sur ce co-processeurs n'est pas de mise dans cette rubrique. Mais le mot est passé à Loïc Far, qui a promis d'aborder le sujet très prochainement dans sa rubrique Hardware.
Attacher deux sprites l'un à l'autre ne pose théoriquement aucun problème, pour peu que l'on respecte les quelques règles suivantes :
1) on ne peut attacher les sprites que par paire : spr1 à spr0, spr3 à spr2, spr5 à spr4 et spr7 à spr6.
2) le bit ATTACH (bit 7 du deuxième mot de contrôle dans la structure sprite) doit être positionné pour le second sprite (celui de numéro impair : 1, 3, 5 ou 7).
3) les coordonnées des deux sprites attachés doivent coïncider exactement.
Le Blitter est globalement plus rapide que le 68000 dans 95% des opérations. Dès qu'un transfert avec opération(s) logique(s) est nécessaire, le Blitter enfonce (et de loin !) ce bon vieux 68000, surtout si le bit 10 de DMACON (BLTPRI) est positionné. Par contre, pour de tout petits transferts ne requiérant aucune opération logique, le temps d'initialisation des registres du Blitter par le 68000 rend préférable l'utilisation de ce dernier.
Il n'existe pas de routine "exacte" pour attendre le VBLANK ; le meilleur moyen est de tester le registre INTREQR pour déterminer si l'interruption VBL s'est produite.
Un tableau des MinTerms ? Ce n'est pas une mince affaire... Utilisez donc plutôt le diagramme de Venn suivant, tel qu'il est fourni dans le Hardware Reference Manual (voir figure).



Voici comment l'utiliser.
1) Pour sélectionner une fonction D=A, repérez les numéros de bits totalement inclus dans le cercle A. Ce sont 7, 6, 5 et 4. Mis à 1 dans les MinTerms, ces valeurs donnent :

Bits : 7 6 5 4 3 2 1 0
MinTerms : 1 1 1 1 0 0 0 0 = $F0

2) Pour sélectionner une fonction qui est la combinaison de deux sources, repérez les numéros de bits qui appartiennent à l'intersection des deux cercles concernés. Par exemple, pour la fonction AB (A "and" B) :

Bits : 7 6 5 4 3 2 1 0
MinTerms : 1 1 0 0 0 0 0 0 = $C0

3) Pour sélectionner une fonction qui est l'inverse d'une source (par, exemple a), repérez tous les numéros de bits qui ne font pas partie du cercle concerné. Dans le cas de a, nous avons :

Bits : 7 6 5 4 3 2 1 0
MinTerms : 0 0 0 0 1 1 1 1 = $0F

3 bis) Pour sélectionner une fonction qui est la combinaison de deux sources dont une inversée, repérez les numéros de bits qui ne font pas partie de la source inversée, mais seulement de la seconde. Par exemple, pour la fonction aB :

Bits : 7 6 5 4 3 2 1 0
MinTerms : 0 0 0 0 1 1 0 0 = $0C

4) Pour combiner plusieurs fonctions, effectuez un "or" entre ces fonctions. Par exemple, pour AB+BC :

Bits : 7 6 5 4 3 2 1 0
AB : 1 1 0 0 0 0 0 0 = $C0
BC : 1 0 0 0 1 0 0 0 = $88
AB+BC : 1 1 0 0 1 0 0 0 = $C8

Voilà pour ce diagramme, qui est, comme vous aurez pu le constater, très simple d'emploi.

Ainsi se termine notre Requester du mois. Le mois prochain sera, j'espère, un peu plus fourni en lettres et plus varié.

ABONNEMENTS A L'ANT

Vous êtes déjà abonné ? Ou ne tenez-vous entre vos mains pleines de doigts, que le résultat du travail clandestin de la photocopieuse au patron du père du fils de la concierge ? Si c'est le cas, vous avez encore une change de vous rattraper, on vous en voudra pas, c'est promis-juré.
Car s'abonner à l'ANT, c'est non seulement s'assurer de faire définitivement partie d'une nouvelle race de privilégiés, mais c'est aussi se garantir une source permanente d'informations techniques que vous ne trouverez nulle part ailleurs, en clair comme en codé. Nos journalistes, réputés pour leur sérieux professionel et leur profonde connaissance du monde de l'Amiga, se décarcassent chaque mois pour vous offrir le meilleur journal dédié à la programmation sur Amiga de l'univers, banlieue comprise (1).
Et pour vous aider à progresser encore plus dans votre passion, la disquette accompagnant chaque numéro de l'ANT contient non seulement tous les sources et exécutables de tous les programmes publiés dans le magazine, ce qui est bien la moindre des choses je vous l'accorde, mais elle fourmille (2) en plus des meilleurs utilitaires du domaine public en la matière.
N'hésitez plus ! Abonnez-vous dès maintenant si ce n'est pas déjà fait (et même si c'est déjà fait, d'ailleurs, y'a pas de raison) et profitez de notre offre exceptionnelle de la rentrée : 11 numéros de l'ANT au prix incroyable de 11 numéros de l'ANT !

Et si vous doutez encore du bienfondé de notre proposition, nous tenons à votre disposition des centaines de témoignages verbaux et quelques autres écrits, dont voici in-extenso certains passages choisis au hasard parmi les plus poignants.

"Chère ANT. Si je vous écris aujourd'hui, c'est pour informer vos lecteurs du véritable changement qui s'est produit en moi grâce à vous (...). Avant de vous connaître, j'avais du mal à aborder certains thèmes particuliers, qui dérangent toujours en société : Exec, bibliothèques, multitâche... J'étais mal vu de mes petits camarades de jeu et pour tout dire, cela s'en ressentait réellement sur mon moral. Mais depuis que je me suis abonné, ma vision des choses a changé, je ne suis plus du tout le même. La verve de vos journalistes, le courage avec lequel ils osent parler de la programmation système ont fait de moi un autre homme. A tel point que j'en ai formatté ma disquette avec tous les sources en K-Seka (dont j'étais pourtant si fier !) et que je pense actuellement très sérieusement à acquérir un compilateur C (...).
Peut-être mon cas n'a-t'il rien d'extraordinnaire pour vous, habitués que vous êtes à sauver ainsi les âmes sombres. Mais pour ce que vous avez fait pour moi, je tenais à vous remercier publiquement et devant tout le monde, afin que votre action dure le plus longtemps possible.
PS : Et si certains de vos lecteurs ont encore des doutes après avoir lu cette lettre, je tiens à leur disposition des photos de moi avant et après l'ANT, qui leur prouveront sans conteste toute la véracité de mes dires." (M. Roland G., Paris)

"Chers amis de l'ANT. Ma lettre ne va sans doute pas vous surprendre outre mesure, mais je me devais de vous remercier pour tout le travail prodigieux que vous accomplissez chaque mois. Sans vous, ma vie serait bien morne, dans ma cité HLM. Vous êtes le soleil qui vient mettre un peu de lumière dans cette sombre et triste existence qu'est la mienne." (M. Franck L., Bordeaux)

Vous le voyez, ces témoignages, parfaitement authentiques, démontrent avec certitude la parfaite qualité du service que vous apporteront l'ANT et sa disquette. Abonnez-vous avant qu'il ne soit trop tard (3).
(1) Franchement, même si vous n'en avez rien à foutre de la programmation, rien que ça, ça vaut la peine de les encourager. (2) Jeu de mots ! (3) Pour vous, bien sûr.

Bon à découper et/ou à photocopier et à retourner de toute urgence, accompagné de votre règlement par chèque bancaire ou CCP, à : **Amiga VPC - Service Abonnements ANT - 16, Quai Jean-Baptiste Clément - F-94140 Alfortville**

Oui, je décide moi aussi de ne pas mourir idiot, et je m'abonne à l'ANT. Je profite de votre offre exceptionnelle de la rentrée de 350 F les 11 numéros seuls, ou 600 F les 11 numéros et leur disquette, soit aucun numéro gratuit. Je suis bien conscient d'avoir une chance de pendu et qu'une telle occasion ne se représentera sans doute jamais dans ma vie.

Je m'abonne pour 11 numéros seuls ☐

Je m'abonne pour 11 numéros avec leur disquettes ☐

Non, je ne m'abonne pas tout de suite à l'ANT, je tiens d'abord à tester par moi-même. Parce que vous comprenez, moi, les témoignagnes bidons, je connais, hein (on m'la fait pas, j'suis avocat). Alors je vous commande les anciens numéros signalés ci-dessous, au prix unitaire mais tout aussi incroyable de 45 F pièce.

Anciens numéros souhaités : ..... pour un total de ..... F.

TRANSACTOR

CONDITIONS GENERALES
1.) La rubrique Transactor de l'Amiga NewsTech est la rubrique écrite par les lecteurs pour les lecteurs. Son but est de leur permettre de s'exprimer librement sur tout sujet qui les passionne, dans quelques langage de programmation que ce soit. Seule restriction : la Rédaction doit avoir la possibilité de tester le bon fonctionnement de tous les programmes fournis, complets ou simples routines d'illustration.
2.) Tout lecteur désirant participer à la rubrique Transactor devra envoyer son article sur disquette au format Amiga, en respectant les quotas définis aux paragraphes 4 et 5. Les disquettes ne seront pas retournées, sauf demande expresse et écrite de l'auteur.
3.) Tout article publié sera rémunéré au tarif forfaitaire de 1, 500 F HT les deux pages ; un article de plus de deux pages sera publié en plusieurs fois, à concurrence de trois épisodes maximum. La paiement a lieu le mois suivant celui de la parution. Les auteurs des articles retenus pour parution seront directement prévenus par téléphone.
4.) Les articles doivent parvenir au format ASCII standard (utilisez un éditeur de textes ou l'option de sauvegarde "text only" ou "ASCII" de votre traitement de texte). Les programmes illustrant les articles, qu'ils soient complets ou de simples routines d'exemple, doivent également être fournis sous forme ASCII, et éventuellement en format "brut" pour les langages en disposant (Basic, AMOS...). Le(s) fichier(s) exécutable(s) doivent se trouver sur la disquette. L'auteur doit fournir à l'ANT la possibilité de recompiler ou d'interprêter ses programmes sans difficulté (joindre au besoin une version "run-only" du langage utilisé).
5.) La taille totale du fichier ASCII donne le nombre de signes de votre article ; une page de l'ANT contient en moyenne 7, 000 signes de texte. Une colonne de listing contient 80 lignes de 65 caractères de large. Un programme de 160 lignes occupe donc une page entière. Les programmes en assembleur particulièrement longs peuvent être publiés en trois colonnes de 40 caractères chaque (240 lignes par page).
6.) Toute proposition d'article pour Transactor devra être accompagnée du bon ci-dessous, découpé ou photocopié, mais en aucun cas reproduit à la main.
7.) Toute proposition d'article pour Transactor entraîne l'acceptation par l'auteur que les programmes joints soient placés sur la disquette d'accompagnement de l'ANT. L'auteur doit préciser dans des commentaires en tête de listing s'il place son programme dans le domaine public ou s'il désire conserver son copyright.
8.) Ni l'ANT, ni Commodore Revue SARL, ni les auteurs collaborant à la rubrique Transactor ne sauraient être tenus pour responsables en cas de dommages quelconques survenus à l'ordinateur, suite à une mauvaise utilisation des documents (textes et programmes) publiés dans cette rubrique.
9.) Toute proposition d'article pour Transactor entraîne l'entière acceptation par l'auteur de ce règlement.

PROPOSITION D'ARTICLE A retourner à : Amiga NewsTech (Transactor) - 5, rue de la Fidélité - F-75010 Paris - France

Je soussigné,
Nom: ..... Prénom: ..... Tél : .....
Adresse : .....
Code Postal : ..... Ville: ..... Pays: .....
déclare être l'auteur de l'article et du (des) programme(s) ci-joints. Je désire les soumettre à l'appréciation de la rédaction de l'Amiga NewsTech pour une éventuelle publication, dans le cadre de la rubrique Transactor.

Sujet de l'article : .....
Taille du fichier texte : ..... signes - Taille du (des) listing(s) : ..... lignes
Remarques complémentaires: .....
.
Je déclare avoir pleinement pris connaissance du règlement ci-dessus et l'approuver entièrement. Signature (précédée de la mention "lu et approuvé") :