AMOS NEWSLETTER
ISSUE 1 • AMOS User Club

WELCOME!

Hello and welcome to the first official AMOS Newsletter. Included in this issue are articles on AMOS programming from the absolute beginner upwards, including yet another 10-line game. There's also an article by the (almost) world famous Professor Speck O.D.D. on the benefits of using Pseudophysics in game design.

Remember, this is a USER club, so send in your articles, views and ideas for the newsletter. We rely on your skills to add new ideas to the magazine.

Even total beginners can come up with good programming ideas, so get writing. Please send all contributions as either text files on a disk, or as a printout. If your handwriting's anything like mine it will be unreadable.

Send your contributions to the normal club address:

1 Lower Moor, Whiddon Valley,
Barnstaple, North Devon. EX32 8NW

The Helpline

If you get totally stuck while programming in AMOS then feel free to phone the AMOS helpline here at Shadow Software. The number is 0271 23544.

Remember to state that you are programming in AMOS (as the STOS helpline is on the same number). Later in the year, each helpline will get its own number to make life easier.

The best time to ring is between 2pm and 7pm UK time (please check what time it is in the U.K if phoning from overseas. Being woken up at 5am is not funny !) Please remember to have your membership card handy when you phone the helpline, as you won't be able to use it without a valid number.

The AMOS Newsletters will be released roughly every two months starting from July. Sometimes they might get a little delayed due to unforseen circumstances (like no articles coming in!), but this effectively means that you get more of a chance to use the helpline before you re-subscribe next July/August!

If you have joined after July, you may have received a batch of newsletters, including this one, in one go when you joined. This is because the club's subscriptions run from July to August(ish) every year.

Anyway, I shall leave you to enjoy reading this newsletter, with thanks to all the staff at Mandarin for their help and Sandra Sharkey for running the PD Library.

Aaron Fothergill

Inside this issue:

Letters 2
How I wrote Cartoon Capers 4
Public Domain Library 7
A500 Blues 8
Adding to Magic Forest 10
Absolute Beginners 12
The Art of Pseudophysics 15
Bobs, Sprites &other Lifeforms 17

Letters

I am writing to you as I am having some problems with AMOS.
1) When I load a sprite from the Sprite Editor or the Sprite Grabber into AMOS Basic, it doesn't have the right colours. The Get Sprite Palette command changes the background and sometimes the sprite colours, but the sprite is not displayed in the correct colours. But when I load a sprite that is already on the disk, like the monkey, it has the right colours. When I load one of my own sprites back into the Sprite Editor it is displayed in the correct colours in the Sprite Editor only.
2) Is it possible to load two IFF pictures and then display one after the other without loading in between?
3) This bears some relationship to problem 2. I am having some problems with hardware scrolling (software scrolling is too slow or too jerky). I would like to be able to develop a detailed scrolling background that doesn't just repeat itself, a bit like the one found in Xenon II. This background changes as you go along. I would like to be able to line up five or six IFF pictures vertically or horizontally and scroll over them. Could this be done without deadlines?
Philip Deakin, Notts

1) The problem you are having with displaying sprites in their proper colours probably stems from confusion between sprites and bobs (See this issue's article).

You must remember that sprites use a different palette. To display them in their correct colours you will probably find it easier to use the BOB commands instead of sprite commands (also make sure you are not doing any FADE or GET PALETTE commands after the GET SPRITE PALETTE).
2) To load a couple of IFF pictures and then display them, use the AUTOVIEW OFF command to switch off autoview (thus screens are only displayed after every VIEW command rather than under interrupt). Then load your screens into separate screen numbers. You don't have to set up the screens as LOAD IFF does this for you. Once the screens are loaded, simply use:

```
SCREEN TO FRONT n : VIEW
```

where *n* is the number of the screen you want to view. If the screens are different sizes, and some of the underlying screens show through, use the SCREEN DISPLAY command to make it so that none is displayed. e.g:

```
SCREEN DISPLAY n,,0,,1.
```

This means that the one line of the screen will be displayed, and that will be off the top of the actual screen area. To view it use:

```
SCREEN DISPLAY n,,48,,screen height
```

then the:

```
SCREEN TO FRONT n : VIEW
```

commands. The 48 is the hardware screen co-ordinate of the top of the screen. You can move it up or down by changing this number.
3) Scrolling is always a problem! The following program shows you how to scroll a screen using software, and proves that it can be done both smoothly and with reasonable speed. I will be covering hardware scrolling next issue. You could modify the following program so that it scrolls multiple screens. The trick lies in modifying line 23 or adding to it, so that it adds lines from another screen, rather than copying the left of the current screen. There will be a full article on scrolling next issue (I'm going to need some time to go over the various ways of scrolling an AMOS screen!) As with all the listings in this issue, don't type in the line numbers, they are for reference only. This program uses the AMOS Data disc, so have it ready in the drive.

```
1 Hide On : Curs Off
2 Dir$="AMOS_DATA:MAGIC_FOREST"
3 Auto View Off
4 Load "BACK1.ABK",5
5 Load "MFSPRITES.ABK"
6 Unpack 5 To 0
```

```
7 Screen Open 1,336,200,16,Lowres
8 Curs Off
10 Get Palette 0
11 Screen To Front 0
12 Screen Display 1,,0,,1
13 Screen 0
14 Double Buffer
15 SC=2 : Rem scroll speed (Max 16)
16 X=60 : Y=196 : Rem bob location
17 Set Bob 1,-1,, : Rem switches off background drawing for bob 1
18 Rem ALL bobs to be displayed in this routine MUST have their background'
19 Rem drawing turned off ! '
20 Update Off : Rem switches off bob interupts
21 While Mouse Key=0
22 Rem scroll screen 1 at scroll speed SC '
23 Screen Copy 1,0,0,SC,200 To 1,320,0
24 Rem to scroll multiple screens together, just change the above line to add the '
25 Rem next chunk of screen to the right of screen 1'
26 Screen Copy 1,SC,0,336,200 To 1,0,0
27 Rem display it on screen 0 '
28 Screen Copy 1 To Logic
29 Rem do your bobs here !'
30 Bob 1,X,Y,7+T
31 Update : Rem this displays the bob(s) and does the screen swap for you
32 View : Rem displays the screen
33 Inc T : T=T mod 3 : Rem increment bob movement (loop it from 0-2)
34 Rem joystick movements (left/right, up=jump) '
35 If Jleft(1) and X>0 Then X=X-2
36 If Jright(1) and X<160 Then X=X+2
37 If Jup(1) and Y=196 Then JMP=-16
38 Rem jump routine '
39 If JMP or Y<196
40 Y=Y+JMP
41 Inc JMP
42 If Y>=196
43 JMP=0
44 End If
45 End If
46 Wend
```

■

Mandelbrot Generator

The following is a Mandelbrot Generator sent in by John Findlay of Braunston, Northants. It takes about 45 minutes to complete a full page, but change SCRY# to 56.0 and SCRX# t o 60.0 to try out different magnifications on a mini brot (2-3 minutes).

This is a full mandelbrot, meaning the whole of the image is generated. More interesting effects can be seen by magnifying and/or shifting the image. To do this change the values of XMIN, XMAX, YMIN and YMAX. Try:

```
XMIN#=-2.01
XMAX#=-0.55
YMIN#=-0.52
YMAX=0.55
```

Thirty two colours can be used; add the colours to the screen and change *K* to 31. When typing in this listing, the line numbers are just for reference, don't enter them.

```
1 Screen Open 1,320,256,16,Lowres : Curs Off : Flash Off : Hide : Ink 0 : Bar 0,0 To 320,256
2 Palette $0,$B90,$AA0,$9B0,$8C0,$7D0,$6E0,$5F0,$F0,$E0,$D0,$C0,$B0,$A0,$90,$0
3 SCRY#=256.0 : SCRX#=320.0 : K=15 : XMIN#=-2.01 : YMIN#=-1.2 : XMAX#=0.55 : YMAX#=1.2 : H#=(XMAX#-XMIN#)/SCRX# : V#= (YMAX#-YMIN#)/SCRY#
4 For Q#=0.0 To SCRY# : For P#=0.0 To SCRX# : M#=XMIN#+P#*H# : N#=YMIN#+Q#*V# : I=0 : X#=0 : Y#=0
5 LABEL:
6 W#=X#*X# : Z#=Y#*Y# : R#=W#+Z# : Y#=2*X#*Y#+N# : X#=W#-Z#+M# : I=I+1
7 If R#<4 and I<K : Goto LABEL : End If : XU=P# : YU=Q# : Ink 3 : Plot XU+1,YU : Ink I : Plot XU,YU
8 If Mouse Key=2 : Edit : End If
9 Next P# : Next Q# : Repeat : Until Mouse Key=1
```

■

How I wrote

by Simon Cook

IIt has given me great pleasure to have been involved in the development of AMOS over the last 12 months. In that time I have seen the amazing possibilities promised by AMOS become reality. In fact, it has surpassed all expectations, to become the most significant advancement in the production of home computer entertainment software in years. At last, true commercial quality games can be written in a high-level structured language, rather than 68000 assembly code.

Due to the pure speed and power of AMOS, no longer is a complicated, technical knowledge of the internal workings of the Amiga needed; the design, quality and playability of the game itself become the primary consideration in development. This can only result in the appearance of many top-quality, playable games, produced by people who do not have the time or inclination to master 68000 assembly language, but have many fantastic ideas.

So, how exactly do you go about writing a commercial quality game?

First of all, you need an idea. This may be a completely original concept, or merely an enhancement of a tried and tested game style, as was the case with *Cartoon Capers*. I took the classic beat-'em-up style and introduced cartoon humour.

The usual oriental martial arts experts were replaced with a cartoony dog and cat, and the chops and kicks given humorous touches. Then, classic cartoon traps and weapons were added and the whole concept began to take shape. I imagine that since you've bought AMOS, you will have plenty of ideas for the type of game you would like to develop.

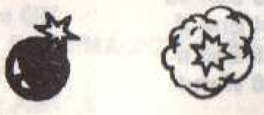
So let's move on to the next stage: Planning or design.

The design stage is the most important part in the development of any piece of software, as it very hard to progress without some idea of exactly what you want to do.

Note that not every last detail need be considered here, as many new ideas will probably come to you as you progress with the game. However it helps to have a good, solid foundation to start from. Items such as the size and number of sprites/bobs, layout of the screen(s), general rules for the game, memory considerations and so on, need to be thought out and planned on paper.

Test graphics can also be designed on the computer. The graphics at this stage do not have to be of commercial quality, just the right size and basic shape; you can touch them up and add extra details later on in the development.

One good way I have found of producing characters or sprites/bobs, such as the main dog and cat in *Cartoon Capers*, is to draw each sprite/bob the full size of the screen and then use a graphic package to reduce them to the required size. The rough reduced pictures can then be used as working graphics, and touched up at a later stage. Background screens, too, need only be rough outlines at this stage.



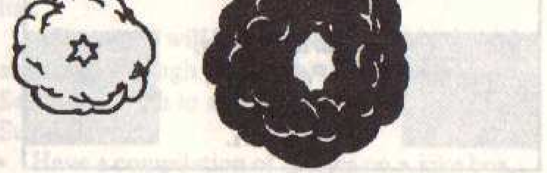
So now you've got an idea, a good basic design, and some working graphics. It's time to start to make things actually move on the screen. The beauty of AMOS, and in particular its animation sub-language, AMAL, is that you can actually get a reasonable idea of how the game will look with only a few lines of Basic. About 10 lines of AMOS Basic, and a few AMAL strings, is all that's needed to display a background and have a couple of bobs animating and moving around the screen. After you've experimented a little, and impressed yourself with the power of AMOS, the basic shell for a game may be written.

All games have basically three parts: A title page/demo, a main game loop, and an ending sequence/high-score table.

The title page and ending sequence can be anything you like. Try using some of the built-in features of AMOS to create spectacular demos. In fact, this is a good way to learn the capabilities of AMOS before actually embarking on the main game loop.

The main game loop usually consists of the following procedures (procedures, of course, are one of the major features of AMOS):

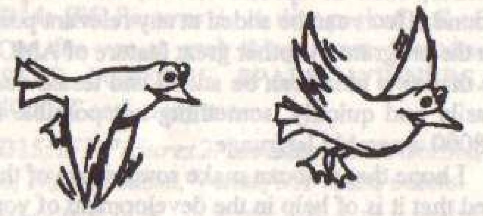
```
Initialise variables, bobs, sprites, screens, etc.
REPEAT Accept new move for each player
IF old AMAL string finished
Get new AMAL string corresponding to new
move
END IF
Check collision detection
Check gameover conditions
Update scores, timer, etc
Update bobs/sprites, etc
UNTIL gameover flag set
End game sequence
```



The initialise section should do things such as load screens, sprite banks, samples and so on, if needed; initialise scores, timers and the like; and set the initial positions for sprites/bobs and screens.

After that, the main REPEAT...UNTIL loop executes until a *gameover* flag is set following some condition such as time running out or a life being lost.

During this loop a number of things occur: First, a player's move is accepted either from a joystick, the keyboard, or an intelligence routine in the case of computer-controlled players. Then the corresponding AMAL string for that move should be fetched from memory and stored in the current move string for that player; ie. each



possible move that a player may make should have a corresponding AMAL string stored somewhere in memory.

An AMAL register should be used to indicate when a move is started and when it is finished. When a move is finished, further inputs may be accepted and new moves made.

Next, any collision checking should be done. If a collision occurs which will change the current move a player is making, for example, an explosion, the current move string should be replaced with the new collision string without checking whether the current move has finished or not. That is, the player's move is overridden by the new collision move.

Then the conditions for a gameover situation should be checked, and if they occur the gameover flag should be set. Next the score board, timer and so on, should be updated, if necessary. Finally, the player's current move is performed by moving it into AMAL, using something like:

```
IF currentmove$ <> "" THEN Amal
channel#, currentmove$
currentmove$ = ""
```

Any other effects, such as the traps and other characters in *Cartoon Capers*, can be called and moved prior to the collision detection checking procedure. They can also be updated in the same way as the players.

That summarises the basic shell for a game and the major areas which need to be programmed. Extras, such as special effects, and sampled sound effects can be added at any relevant point in the program. Another great feature of AMOS is that new ideas can be added and tested very easily and quickly, something not possible in 68000 assembly language.

I hope that you can make some sense of this and that it is of help in the development of your own games, and look forward to playing some great AMOS games very shortly. In the meantime, go out and buy *Cartoon Capers* to see what exactly AMOS is capable of, and see if you can do better!

Cartoon Capers will be released in October. ■

Public Domain Library

What it's about

Listed on this page are the first discs to enter the AMOS Public Domain library. We intend to fill the library with useful utilities, graphics, sound resources – in fact anything to do with AMOS. With AMOS 3D on the horizon we plan to supply additional 3D object data discs through the library.

We also need your support. Over 90% of the PD discs in the STOS Club are supplied by members. By sharing your creations with other AMOS users everyone will benefit and it will help to spread the word about AMOS.

How it works

Each disc costs just £2.50 each – or £1.50 if you supply your own disc. If you order three discs or more, deduct 20p per disc (including the first three). For example, four discs will cost £2.30 each (£9.20 in total). Please add 25p postage per disc if you live in Europe – or 50p per disc if you live elsewhere in the world, otherwise your discs will be sent by surface mail!

Each disc has a specially-printed AMOS Public Domain sticker kindly produced by Mandarin Software so your PD discs will match the rest of your AMOS master discs.

Ring Sandra on 0942 495261 to find out about the latest public domain titles to be added to the library.

If you have any programs which may be suitable please send them along (preferably as unprotected Basic files).

Send cheques, postal orders or stamps to:

Sandra Sharkey,
25 Park Road,
Wigan,
WN6 7AA.

Public Domain Library

APD1: *GMC (Games Music Creator).* A powerful and easy-to-use utility that allows you to create musical tunes. Uses all four channels and comes with DOC files on disc. Use the GMC-to-AMOS converter (on the AMOS program disc) to convert the GMC music data files into the AMOS music format, then play it back using the MUSIC command.

APD2: *Treasure Search.* Hunt for treasure in this clever educational co-ordinates game by Peter Hickman, the author of Number Leap. It includes excellent graphics and sampled speech. This disc contains RAMOS, the Run-only version of AMOS (as found on the EXTRAS disc).

APD3: *Fonts Disc 1.* This contains 14 typefaces including Times, Bookman,Helvetica and Tiny.

APD4: *Fonts Disc 2.* 13 including Avant Garde, Celtic, Palo Alto, Bassel, Peignot and Aldous.

APD5: *Fonts Disc 3.* 14 including Broadway, Camelot, Future, Stencil and Vancouver.

APD6: *ST-Amiga Disc.* Allows you to read ST discs on the Amiga, so you can transfer your programs from STOS-AMOS (in ASC format). Also includes the STOS-AMOS sprite converter and sample bank converter. Note: Because of the way this program works, it effectively needs two disc drives.

APD7: *Virus X 4.0.* The Amiga's most widely used virus killer. We'll supply the newest version as it becomes available.

APD8: *Music Tunes.* A collection of tunes by Alastair Brimble, specially commissioned by Mandarin for AMOS.

APD9: *AMOS Big Demo.* The demo written by Peter Hickman to show off AMOS. Although written originally on an early version of AMOS, it is still quite impressive!

APD10-13: *Amiga Samples 1-4.* Peter Hickman's useful collection of IFF samples for use in GMC, Soundtracker, Sonix or AMOS itself.

APD14: *IFF Pictures 1.* A disc packed full of clip art for you to grab things from, includes CASTLE, PORSCHE, SPARK, WINNERS, KINGTUT and many more.

APD15: *IFF Pictures 2.* Includes Faces, Gorilla, Odie, Ninja, Mazda, Venus, MTV and others.

APD16: *IFF Pictures 3.* Compact disc, Walkers, Trex, Kodak, Captain Kremlin, Micky......

APD17: *92 samples ready for use in GMC.*

APD18: *76 more samples for GMC compositions.*

APC19: *Microman's Music Sensitive Balls Demo.* Pretty graphics and a real nice environment allow you to play back the tunes from the AMOS package. Due to its size A500 half meg users will only be able to play small musics.

MEGA MUSIC **£150 PRIZE** *COMPETITION*

We're looking for the best Soundtracker, GMC or Sonix Music demos written in AMOS.

You can add graphics and animation to enhance the atmosphere created by your music – be creative, use sprites and bobs and the VU meter commands to bring that composition to life.

All entrants will be made Public Domain and available through the club. You have until September 30th to send in your entry.

Suggestions:

- Have a compilation of musics on a juke box, with the ability to choose a particular piece of music to play.
- Animate in time to the beat.
- Add user control to your demo – joystick, mouse or keyboard.
- Make it fast and furious fun to watch.

Post your entries to Richard Vanner at Mandarin Software. The best entry will receive a cheque for £150 courtesy of Mandarin.

Please send in your creations for the PD Library. We need games, demos, music, graphics, utilities in fact anything! ■

A500 Blues

by Aaron Fothergill

If you are working with AMOS on an unexpanded A500, you will no doubt be very happy to hear that I will be doing a series of articles on how to get as much as possible out of limited memory. Those of you working with one meg should also read these articles – if you are going to release something commercially, it should fit into a half-meg machine!

General Tips

• Make sure that AMOS is trying to close the Workbench when booting. This saves about 40k or more and is set on the EDITOR OPTIONS menu selection in the CONFIG.AMOS program on your master disk.

• If you can bear not using it, switch off or disconnect your second drive before switching on the Amiga and booting AMOS. This can save 20k or more.

Pruning the Sprite Editor

You might want to fit more or larger sprites into the Sprite Editor. This can be done by pruning off bits you don't need. If you never use the Niceness routines (screen and button appearance and Airbrush speed), you can remove these and save about 7k. If you are only working in one resolution you can remove the resolution selector gain another 1k. Now 8k might not seem like a lot, but on average you can get another 30 or so sprites in memory! When modifying the Sprite Editor, or any of the programs on your master disks, work on backups only! **NEVER your master disks!**

Removing the NICENESS routines

• Use the FIND TOP menu to search for **Gosub NICENESS.** The statements around it should look like: **If C=11 Gosub NICENESS End If.** Delete these three lines from your program.

• Then search for the subroutine which is called **NICENESS:** (note there is NO space between the S and thecolon). Mark this line (1182) with the BLOCK START option in the BLOCK menu

• Search for **SELREZ:** and mark the previous line (it should be 1288) with the BLOCK END option from the BLOCK menu.

• Do a CUT BLOCK from the block menu, thus removing the Niceness routines.

• This will leave a few procedures that are now redundant, so using FIND and CUT BLOCK remove the following: **SMALLBUTTON, TWINBUTTON, QUADBUTTON.**

• If you have a suitable art package, you can get more memory back by the following:

a) Go to direct mode and save the menu buttons picture with the following lines:

```
Default Unpack 6 to 0
Save IFF "MENU_BUTTONS.IFF",0
```

then after saving your modified version of the Sprite Editor on a spare disk, load up your art package, load in the MENU_BUTTONS.IFF file in 8-colour Low Resolution Mode and procedure to erase the Shadow Software logo to the bottom right of the icons, and the Blue, Light Blue and Yellow Buttons. Then resave the picture under the same name.

b) Re-boot AMOS (after first switching off)

c) Enter the following program:

```
Load Iff "MENU_BUTTONS.IFF",0
Spack 0 to 6,0,0,320,144
Save "MENU_BUTTONS.ABK",6
```

d) Run this program, then load in your pruned sprite editor, go to direct mode and type:

```
LOAD "MENU_BUTTONS.ABK",6
```

and your sprite editor should have an extra 8k or so of memory over the original.

Removing the Resolution Selector

• Find the **SELREZ:** subroutine and its Return statement, mark it as a block and remove it with Cut Block.

• Find **Gosub SELREZ** and remove the surrounding **If....Endif** (five lines). To work in a different resolution, change line 18 which sets the *REZ* variable to the default resolution. Values from 0 to 7 are valid and represent:

0= Low res 8 colours
1= Low res 16 colours
2= Low res 32 colours
3= Low res 64 colours (EHB)
4= Hi res 2 colours
5= Hi res 4 colours
6= Hi res 8 colours
7= Hi res 16 colours

Once finished, save your modified Sprite Editor as SMALL_SPRITE.AMOS on a spare disk. Reboot AMOS or change the buffer size back to normal and load this version.

Another way of saving memory, is to make the Menu Buttons screen load in from disk at the start of the program. This is slightly less convenient, because you make sure that the disk with the MENU_BUTTONS.ABK file is in the current drive whenever you run the Sprite Editor, but it does save you nearly another 7k of memory!

To do this modification:

• Go to direct mode and type:

```
Save "MENU_BUTTONS.ABK",6
```

• Now return to the editor and find the procedure **TITLEBAR**

• You will have to unfold this procedure. Once done, check about seven lines down from the start of the procedure and you should find:

```
Unpack 6 to 2
```

• Insert a line before this reading:

```
Load "MENU_BUTTONS.ABK",6
```

then insert a line after the Unpack line, reading:

```
Erase 6
```

This version will load in the menu buttons, unpack them to the screen and then erase the now unused bank.

Other options include trimming out commands that you don't use (difficult) or simplifying the Menu Buttons screen down to four or even two colours so that it uses less memory. This means checking through the program for any of the routines that draw to the menu screen to update the buttons to make sure that illegal colours are not used.

More on A500s and the sprite editor in the next issue. ■

Stop Press!

• AMOS is still selling really well around the world. Pactronics in Australia is doing particularly well – they've even set up their own AMOS Club.

• French and German language versions of AMOS go on sale at the end of the year – and those people who've already bought the English AMOS can swap their manuals with Mandarin for a couple of pounds.

• AMOS programs are arriving at Mandarin at the rate of up to 10 a day! There's a Worm game, two Breakouts, a duckshoot, a horizontally-scrolling platform game, lots of demos – and more besides. Mandarin have decided to keep the discs, but send a free PD disc to all senders. Contact Sandra to get your mits on some of these (though some may not be for public domain).

• Fun School 3 (Amiga) goes on sale at the beginning of September. It's available for Under 5s, 5-7s and Over 7s with six programs per package and a price of £24.99 – and it's written entirely in AMOS using the RAMOS run-time system. Check it out – it looks amazing! (You can get Peter Hickman's FS3 demo from Sandra.)

• Cartoon Capers has been delayed as it won't quite fit into an unexpanded A500. François is working on a routine to save more space. ■

AMOS Hotline

Membership has its privileges! Once you subscribe to the AMOS Club you will be given a number to ring for on-line technical help. Aaron Fothergill, author of Magic Forest, the Sprite Designer and the Map Editor will be on hand to help you with any AMOS-related subject.

Tel: 0271 23544

Adding a High Score T

Most arcade games have some form of storing the highest scores. In fact in the early days of arcade gaming, getting a high score and getting your name or initials on to the table was the highest goal.

As games developed, the added incentive of finding out what was on the next level took away some of the need for a high score, but they are still nice to have.

The principles behind a high-score table are as follows:

• Check a player's score against the high scores, starting from the highest and working down.
• If the player's score is greater than one of those on the table, move the beaten score and all those below it down one place, get the player's name or initials and store them with his score in the vacated slot. Remember when moving the scores down to also move the relevant names down with them.
• Display the high scores in a table, as a scrolling list, or as a series of bunnies jumping across the screen with banners containing the scores (just a few ideas there!).

The following AMOS lines add a simple high score table to the Magic Forest game on your AMOS Data disk. Make a backup of the Data disk and add them to the program. The lines here are numbered so that you know when a new line starts, but remember to leave out the line numbers.

A) Find the Dim statements at the start of the program and add the following lines after the normal two Dim statements:

(1) Dim HIGH(9),HIGHNAME$(9)
(2) Gosub INITHIGHSCORES B)

Find Line 463 and add the following lines after the Erase 1 line:

(3) Screen Copy Logic To Physic

Then add the following lines after the Loop instruction on line 488:

(5) Rem Initialise high score table
(6) INITHIGHSCORES:
(7) A=0 : Repeat
(8) HIGHNAME$(A)=Left$("Magic Forest" +Space$(20),20)
(9) HIGH(A)=11000-(A*1000)
(10) Inc A
(11) Until A=10
(12) Return
(13) Rem High Score Table
(14) HIGHSCORE:
(15) Rem First check to see if score is higher than the 10 stored
(16) A=0 : Repeat
(17) If SCORE>HIGH(A)
(18) Gosub HIGHADD
(19) A=99
(20) End If
(21) Inc A
(22) Until A>=10
(23) Rem Display high scores
(24) Rem To do this, we will grab the lines as blocks from another screen
(25) Rem So we will need another screen
(26) Auto View Off
(27) Screen Open 2,320,10,16,Lowres
(28) Curs Off : Flash Off
(29) Screen To Back 2
(30) Auto View On
(31) Rem The following procedure displays the text, and grabs it as a block
(32) HIGHLINE["High Scores",12,61]
(33) A=0 : Repeat
(34) H$=HIGHNAME$(A)+Space$(5)+Right$ ("000000"+Mid$(Str$(HIGH(A)),2),6)
(35) HIGHLINE[H$,15,A+62]

able to Magic Forest

```
(36) Inc A
(37) Until A=10
(38) Screen Close 2
(39) Put Block 61,80,10
(40) A=0 : Repeat
(41) Put Block A+62,50,30+A*15
(42) Inc A
(43) Until A=10
(44) Return
(45) HIGHADD:
(46) Screen Open 2,320,25,16,Lowres
(47) Get Palette 1
(48) Curs Off : Flash Off
(49) Set Rainbow 1,3,25,"","","(1,2,8)"
(50) Ink 3 : Bar 0,0 To 320,25
(51) Ink 1 : Box 0,0 To 320,25
(52) Paper 2 : Pen 14
(53) Locate 0,0
(54) Centre "Enter Your Name!"
(55) Rainbow 1,0,Y Hard(1),25
(56) Paper 3 : Pen 15
(57) Locate 0,1
(58) Screen To Front 2
(59) Line Input ":";NAME$
(60) NAME$=Left$(NAME$+Space$(20),20)
(61) B=9 : While B>A
(62) HIGH(B)=HIGH(B-1)
(63) HIGHNAME$(B)=HIGHNAME$(B-1)
(64) Dec B : Wend
(65) HIGH(A)=SCORE
(66) HIGHNAME$(A)=NAME$
(67) Screen Close 2
(68) Return
(69) Procedure HIGHLINE[A$,C,B]
(70) SC=Screen
(71) Screen 2
(72) Locate 0,0 : Paper 0 : Pen C
(73) Print A$;
(74) Get Block B,0,0,Len(A$)*8,10,1
(75) Screen SC
(76) End Proc
```

Once you've typed that lot in, you should have high scores with your game!

Have fun experimenting with the above routines, to modify the high-score table to what you prefer. You might want to store and display the level that the player got to, in which case you would have to set up another array to store it, and remember to move it with the names and scores. You might also want to display the table a bit more dynamically.

My offering is a little ordinary (pink bunnies towing banners?). Oh yes, the golden rule of working on the Amiga:

SAVE IT NOW!!! ■

Drop us a line!

Whether you've got a problem or just something to say, we'd like to hear from you.

We're on the lookout for letters, articles or suggestions to make the Newsletter as exciting and informative as possible. You can use old-fashioned pen and paper, but we'd be delighted to receive Ascii or Protext files together with a printout. Hope to hear from you soon!

Aaron

Absolute Beginners

Number 1

Learner programmers seem to be split into two groups: Those who have programmed in Basic before but not AMOS, and those who have never programmed at all.

For this reason I am going to split this series into two separate articles: ***Absolute Beginners*** will help those who really have never programmed before, and *Almost Beginners* will cover aspects of programming a level up, for those just converting to AMOS or who have never attempted a major program before.

So for our Absolute Beginners, in this issue we are going to write a very simple dice game. This program will introduce some very basic features of programming along with a few handy tips.

There will be one small programming convention here: Because AMOS can have lines longer than we can list on one line in this newsletter, I will be using line numbers for this program instead of labels so that you know when to enter a new line. We will convert to labels in a later issue.

The first question you should ask yourself before writing a program is What am I doing? A program is a sequence of commands that the computer will follow exactly, so you must make sure you get them exactly right. By deciding beforehand what you are going to do in the program and how you are going to do it, you will make life a lot easier.

For our dice game, the rules are simple: The computer rolls a dice to produce a number between 1 and 6. Then the player rolls a dice for another number between 1 and 6 and the highest roll wins. We also want to add a few embellishments such as the ability to enter the player's name, and to play again.

Before writing the program, it can be benificial to work it out in pseudo-code. This is where you write down a summary of what the program will have to do, without having to remember all the commands.

Our program will need routines to:

```
1. Enter player's name
2. Roll the computer dice and display it
3. Roll the player dice and display it (with
   player's name)
4. Check for winner
5. Ask player if he wants another go
6. If so then go back to 2
7. Otherwise stop
```

And that's the game in a nut-shell. As you can see, pseudo-code makes the program look less complicated. Believe it or not, it really is that simple!

The first task on our list is to enter the player's name. We will have to store it somewhere as we will need it later to display with his dice roll. So a **VARIABLE** must be used (see issue 0 for an explanation of what a VARIABLE is). As the name is made up of a string of letters, a **STRING** variable has to be used (with a $ after the variable name). We shall call it *NME$*.

The command required to get an input value from the user's keyboard and store it in a variable is INPUT. The type of value depends on the variables being used for input – for instance you can't input strings into a numeric variable. You can also add a prompt to an INPUT command so that you can tell the user what information is required. So our first line will be:

```
10 Input "Enter your name and press
RETURN ";NME$
```

This will display the message: *Enter your name and press RETURN* with a flashing cursor. The user can then enter his/her name and on pressing Return whatever they typed will be stored in the variable *NME$*.

Next we need to generate some random numbers for the dice throws. As we are only dealing with the numbers 1 to 6, we can use normal INTEGER variables in which to store them.

The function =RND(n) returns a value from 0 to *n*, so to get a number from 1 to 6, we simply use *variable=RND(5)* and add 1 to it. We don't use RND(6) because we don't want the number 0 generated for any throw of the dice.

We will call the variable for the computer's throw *D1* and the player's throw *D2*, so the next two lines will be:

```
20 D1=Rnd(5)+1
30 D2=Rnd(5)+1
```

To display the scores, we simply use the **PRINT** statement, which can output text, numbers or a mixture of both. First we display the computer's score:

```
40 Print "The computer rolls a";D1
```

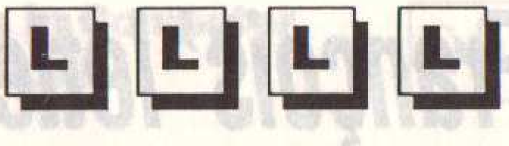
Then we need to output the player's score:

```
50 Print NME$;" Rolls a";D2
```

Notice that on the player's display, we used the variable *NME$* as the first part of the print which already holds his/her name. This means that the name will be output as part of the message.

The only tricky part of the program is working out who wins. To do this, we need to use an IF...THEN statement to work out the argument given to it. If the result is TRUE, it will execute the series of commands after the THEN on the same line. If the result of the condition is FALSE it will go on to the next line (or whatever is after the optional ELSE statement).

First we will check to see if the computer the highest score. This is TRUE when *D1* is greater than *D2*. This can be checked by the line:



```
60 If D1>D2 then print "The computer wins!"
```

Then we check to see if the player has the highest score:

```
70 If D1<D2 then print "The player wins!"
```

Finally we check for a draw:

```
80 If D1=D2 then print "It is a draw!"
```

The meanings of the mathematical symbols used in lines 60, 70 and 80 are:

>	is greater than
<	is less than
=	is equal to

and you can have combinations such as >= greater than or equal to.

All we need to do now is see if the player wants another go. We can use another INPUT statement here and then check to see if the variable used contains *YES*:

```
90 Input "To play again type YES and press RETURN ";Y$
100 If Y$="YES" then goto 20
```

Notice on line 100 how I've used the IF..THEN statement with a string variable and also a GOTO command, which makes the program GOTO the line specified – in this case 20 – providing the result of the test is TRUE.

If you've been typing in the program while reading this, just type RUN, press Return and play the game. You've written your first 10-liner!

LLIST manual bug

The LLIST command, as mentioned in the manual, doesn't exist! To print out your program, or a section of it, mark the area you want to print as a block, then use Print Block from the block menu. ■

François' letter from France

Richard Vanner asked me to write a small intro for the first club's newsletter, and I do it with a great pleasure! It is the opportunity for me to tell you what's in my mind (not much!) at this very moment. As always, I'll ask you to forgive my frenchy English!

THANK YOU! AMOS has been doing really great! You have been wonderful, merveilleux, geniaux.. oops! AMOS has been a success immediately after its release. In fact, we did not wait for such a success: The product was late (thanks to me!), it was in the middle of the Mundiale, and usually, customers don't trust the first version of such a product.

AMOS has reached the second place of Gallup charts, topped only by Kick Off II (a great program!). Now AMOS is still in the top five, and it seems he wants to stay there. The most impressive thing is that you did not wait for the reviews in magazines to buy it. I hope you have not been deceived!

Mandarin have received a lot of letters and registration cards. I can assure you, knowing the customer support team at Mandarin, that every letter is carefully read. They do a sorting, and send me the main ones, and I personally answer to these.

It is a real effort, but it will help us to make AMOS better and better. I am currently working on version 1.2, and this version will include a few new instructions suggested by users. I want to make AMOS as close as possible to your desires (well, the majority!) Bugs bugs. Yes, unfortunately, we have missed a few bugs in AMOS V1.1. During development, AMOS has been tested by 10 persons during several months, but it is nothing compared to 10,000 persons during two months.

Sorry about that, but you must imagine that testing over 1 meg of source code is not an easy (multi) task. The worst thing in big programs is that when you change something in one corner, it can create problem at a very far subroutine that you have almost forgotten!

Let me tell you what happened when I received AMOS master. I was really happy, it was the final point of 16 months hard work. So proud was I when I inserted AMOS language disk in my A500, booted AMOS up, and ... guess what ... pressed HELP! The first key I pressed on my first try of the master was HELP. Bing bang crash! Hello Mr GURU, nice to see you there. It almost killed me!

It was just to tell you that I am aware about bugs, and I do all my best to correct them. On the bug front too, you have been really great. Lots of you have been saying that AMOS was not too bugged compared to other similar products. Thanks!

It may not be too bugged, but it is, so I am currently finishing Version 1.2, where all reported bugs are removed, like:

- HELP
- Global variables in procedure calls
- Accessory crashes
- Disk access problems
- POP PROC with a RETURN in the pile
- and a lot of minor (but annoying) bugs.

Like with STOS, we will release new versions of AMOS by way of a .AMOS program. AMOS interpretor will be, of course, coded in a memory bank, so that people can't extract it without running it from AMOS. Presented like this, it will be public domain, in PD houses and on magazine's cover disk, at NO COST to you! In the near future, I will program a few extensions:

- A DEVICE extension, to add commands like OPEN DEVICE, to have better control over RS232, MIDI and parallel port. Up to now, the Amiga Dos control is really not enough.
- An IFF animation reader, to easily grab animation produced by other programs, and

continued on back page

The Art of Pseudophysics in Computer Games — by Prof A Speck O.D.D.

$\boxed{\Sigma\sqrt{\mu}}$

So you want to write a simulation game eh? Tons of vector maths and physics later you have your game. Unfortunately the gameplay is so terrible you have to offer it with a free T-shirt in order to sell it.

So what went wrong? Simple! Maths and Physics relate to real life. So how many computer space games involve you sitting in a tin can with a vacuum all around you and hordes of slimy things (real ones) attacking you? Answer: None, because the computer can't display things too realistically when its only outputs are sound and a TV screen.

What's really needed is a variation to normal maths and physics, to take into account that your computer world is ever so slightly strange.....

Pseudophysics

Pseudophysics is the application of *Rule of Thumb* in physics, to match the Laws of Physics to a computer enviroment. (For a more accurate explanation of Rule of Thumb and conversion tables to metric and imperial measurements, read *Random Walks in Science*). This means that for your game you must discard any laws of physics that don't look or feel right.

Beware however, there are rumours that the government is bringing in fines for breaking the Laws of Physics. For instance, having a game where people can fly, thus breaking the law of gravity, will carry a maximum fine of £2000. You have been warned! (This doesn't apply yet to overseas readers. However, other governments are looking at this system).

The first thing you must learn about pseudophysics is that nothing is fixed (or rather everything is!) This leads to Speck's 1st Law of Pseudophysics or the Fiddle Factor: *All constants are variable until the game looks and feels right*. This means that you should play around with any of the multipliers in your equations until the game plays the way you want it to and looks correct.

Don't worry if your physics teacher would scream at seeing them. (My physics teacher used to scream at my equations anyway!)

Equations should also be tested out by trying the game on someone else, preferably someone familiar with the style of simulation you are trying. Pseudophysics doesn't discard everything from physics however, this would be silly. It merely uses laws that are useful at the time (much like record companies!).

Sometimes you might have to create a new law to apply to a situation never used before, such as the time taken for a spaceship to get from A to B if it uses hyper-drive (Einstein is now spinning in his grave at 3 revs per minute!) This law might take into account the factor of C being in the way, and the drag factor of a spaceship with planet C wrapped around it.

Sometimes you have to just do a series of extra equations to make up for something that wouldn't otherwise look right. For instance, in the writing of Skystrike and Skystrike Plus, the aircraft is supposed to stall if the airspeed gets too low. To acheive this, a variable *ST* was used. Each frame this variable was subtracted from the altitude of the aircraft.

When the aircraft was flying normally, *ST* was set to 0. When the airspeed got too low, *ST* was set to 10, thus making the aircraft fall out of the sky rapidly. As the airspeed picked up again, the variable was lessened, until the aircraft pulled out of the stall.

Various other routines were then able to use the *ST* variable to determine if the aircraft was stalling, and the speed at which it was dropping (the landing checker for instance). The pseudo-physics routines for Skystrike were developed and tested over two weeks by a team of around eight part-time test pilots, until they were as good as possible.

So remember, real physics just don't work in the computer games world (look at FOFT for an example). So start re-writing those equations. You might want to use the general pseudophysics symbol of ?= which stands for *SORT OF EQUALS* and is very important in psesudophysical equations. For example:

V?= D/T *(Velocity equals Distance over Time multiplied by a Fiddle Factor).*

or:

E?=MC^2 *(Or I'm using hyper space. Eat Ionexhaust Einstein!).*

By the way, this also has the use of making your equations less usefull to anyone who wants to steal your game idea! Have Fun .... Next Issue: How to program in Spaghetti Code.

(Professor Speck is a lecturer in applied Pseudo-Physics at the University of Advanced Gameswriting in Barnstaple. He has also worked as an assistant test-pilot in games such as Skystrike, Skystrike Plus and Yomo. He is currently playing an end of level guardian in YOMO 2389.) ■

Magic Forest 1 Meg bug

There is a small bug in the Magic Forest game which causes the wrong music to be played on the 1 meg version. Line 28 should read:

```
BANK3=1
```

instead of:

```
BANK3=2
```

Mandarin Software is keen to find programmers to write the following applications for possible publication:

- font editor
- adventure game creator
- sampled sound editor to work with most cartridges
- word processor
- database
- spreadsheet
- business graphics
- feature-packed art program
- fractal landscape creator
- ray tracing program
- collections of animated sprites
- original games
- variations on 'classic' games

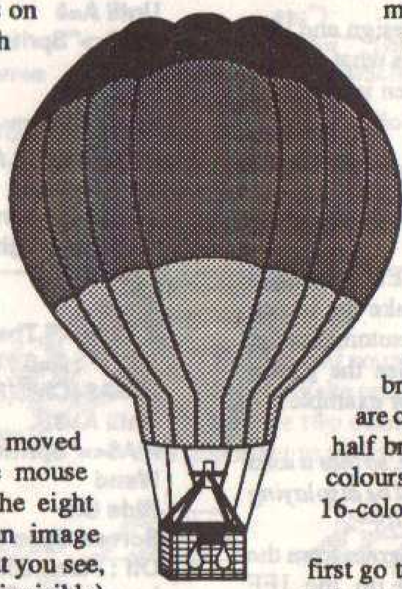
...in fact anything that will appeal to other AMOS users or even non-AMOS users. Mandarin will pay an outright sum or royalties at the standard industry rate.

Please write to Richard Vanner, Development Manager at Mandarin Software in Macclesfield – or ring him on 0625 859333 between 2 and 3pm.

MANDARIN SOFTWARE

Bobs, Sprites and other Lifeforms

Life can be quite confusing at times, especially when a programming language like AMOS has so many different commands, and so many different ways of doing the same job. For instance, AMOS has two ways of manipulating graphical objects on the screen, sprites and bobs. Both these object types have a similar set of commands, and both can be designed from the same program. So what's the difference? This has caused quite a few headaches for some AMOS programmers, so here, in very simple terms, is an explanation of what sprites and bobs are.



Sprites

A sprite is an object that can be moved rapidly about the screen (the mouse cursor is a sprite). Each of the eight available sprites can have an image attached to it (the image is what you see, as the sprites themselves are invisible), with certain limitations. For instance, sprites can only be in either 4 or 16 colours, and are all 16 pixels wide (although they can be any height).

The four parameters in the SPRITE command supply the sprite number, the co-ordinates on the screen to place the sprite and the image number to attach to it. Each sprite can only be displayed at one location on or off the screen at any point in time. So using the following commands:

```
SPRITE 1,10,20,2 SPRITE 1,10,50,3
```

will result in only the second sprite being displayed (as sprite number 1 stops displaying the first image and goes to the new location with a new image). To display two sprites on the screen at the same time, a different sprite number must be used such as:

```
SPRITE 1,10,20,2 SPRITE 2,10,50,2
```

The image number can be the same for all the sprites you are displaying if you want, or it can be completely different. It is just the sprite numbers that must be different.

One tricky point is that sprites use *hardware* screen co-ordinates. So to display a sprite at X=10,Y=20 on your screen, you must convert the X and Y co-ordinates to hardware co-ordinates using the XHARD() and YHARD() functions. For example:

```
SPRITE
1,XHARD(10),YHARD(20),2
```

This isn't the only awkward point – sprites also use their own colour palette. The Amiga supports a palette of 32 different colours (in Extra Half bright mode, the 32 extra colours are copies of the first 32 displayed at half brightness). When sprites use 16 colours they use the second set of 16 (a 16-colour screen uses the first).

So when designing a sprite, first go to 32-colour mode in the Editor and set up the colours you want in colours 16-31, then go back to 16-colour mode and draw the sprite. You can use the SEE AS SPRITE button to check on what the sprite will look like.

Because each sprite can only be 16 pixels wide, you would have problems in displaying larger objects, so AMOS can also use *computed sprites*. Sprites 0-7 are normal 16-pixel width sprites (the mouse cursor is sprite 0), whereas sprites 8 and onwards are computed sprites. With these, AMOS automatically sticks different sprites together, each using a 16-pixel width chunk of your image, so that it is displayed as one image. You have to be carefull however, as there are only eight hardware sprites. This means that you can only use sprites to display upto 128 pixels width of objects at any one time.

Bobs

Fortunately, AMOS can use Bobs (Blitter

OBjects), which are software controlled objects (using the Amiga Blitter chip). They are only marginally slower than sprites, but they have major advantages over them:

- They can be any size.
- You can use as many of them as you want.
- They use the same co-ordinates and colours as the screen.

This makes bobs a lot easier to design and use. What you see in the Sprite Editor is what you get on the screen (assuming the screen is the same resolution and colour set as the Bob).

Tricks with Bobs and Sprites

Q: *How do I stop a bob or sprite flickering on my screen?*
A: By doing a DOUBLE BUFFER command before your display loop, you make the screen display do SCREEN SWAPPING automatically, causing flicker-free animation. See the games Magic Forest and Amosteroids for examples.

Q: *How do I design a Bob or Sprite, so that it uses the same colours as the screen I will be displaying it on?*
A: Load up the Sprite Grabber program from the AMOS disk and run it. Load up the IFF background picture you want the colours from, and grab a sprite from anywhere on the screen. This stores the screen's palette in the sprite bank, which you can now save and load into the Sprite Editor. Once you have designed a few images in the Sprite Editor, you can safely delete the dummy image.

Q: *How do I merge two sprite banks?*
A: The following program will achieve this. Type it in and save it as SPRITE_MERGER.AMOS (not on your AMOS Master disks).

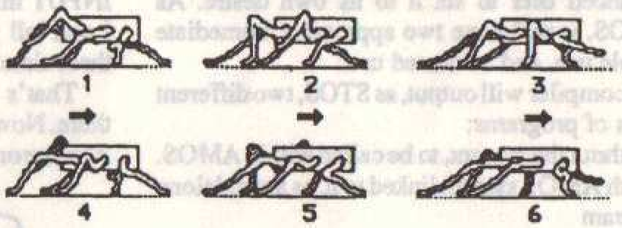
```
' Sprite Bank Merger program
' Shadow Software 17/7/90 written for
' AMOS Club By Aaron Fothergill
'
M=200 : Rem m=max # of sprites in
second bank
Dim XS(M),YS(M),XHS(M),YHS(M),BP(M)
F$=""
While F$=""
F$=Fsel$("*.ABK","","Pick a sprite
bank")
If F$="" Then End
If Upper$(Right$(F$,4))<>".ABK" Then
F$=""
Erase 1
If F$<>"" Then Load F$,1
A$="" : A=0 : Repeat :
A$=A$+Chr$(Peek(Start(1)-8+A)) : Inc A
Until A=8
If A$<>"Sprites " Then F$=""
Wend
F2$=""
While F2$=""
F2$=Fsel$("*.ABK","","Pick another
sprite bank")
If F2$="" Then End
If Upper$(Right$(F2$,4))<>".ABK" Then
F2$=""
Erase 1
If F2$<>"" Then Load F2$,1
A$="" : A=0 : Repeat :
A$=A$+Chr$(Peek(Start(1)-8+A)) : Inc A
Until A=8
If A$<>"Sprites " Then F2$=""
Wend
Hide On
Screen Open 0,320,48,4,Lowres : Curs
Off : Flash Off
Locate 0,0 : Centre "Converting Bobs to
Icons"
Locate 0,1 : Centre "Please wait"
Auto View Off
Erase 2
N=1 : Repeat
A=Sprite Base(N)
XSIZE=Deek(A)*16
YSIZE=Deek(A+2)
HSX=Deek(A+6)
HSY=Deek(A+8)
BP=Deek(A+4)
XS(N)=XSIZE : YS(N)=YSIZE
XHS(N)=HSX : YHS(N)=HSY
BP(N)=BP
Screen Open
1,320,YSIZE+16,2^BP,Lowres
Curs Off : Flash Off
Get Sprite Palette
Screen To Back 1
Ink 0 : Bar 0,0 To XSIZE,YSIZE*16
Paste Bob 0,0,N
```

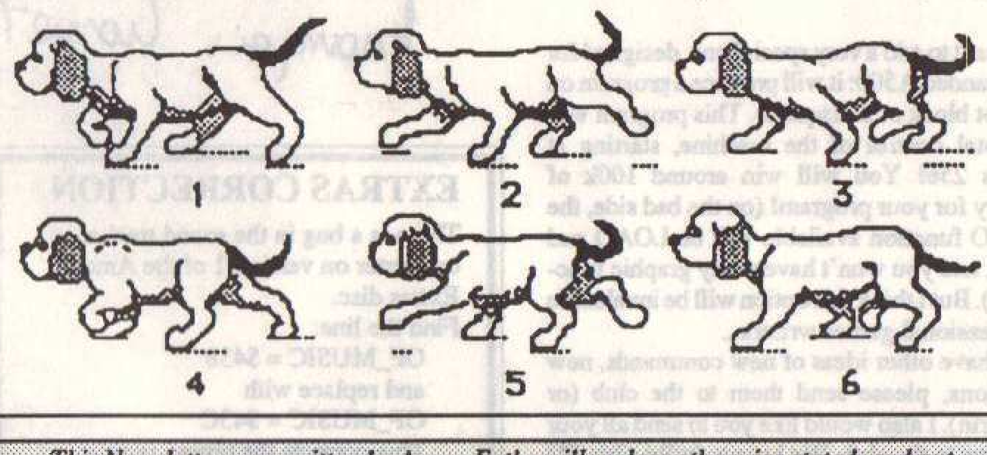
```
Get Icon N,0,0 To XSIZE,YSIZE
Inc N : Until N>Length(1) : L=N
Erase 1
Screen 0
Locate 0,2 : Centre "Now Adding to first
Sprite bank"
Load F$,1
N=1 : Repeat
XSIZE=XS(N) : YSIZE=YS(N) :
HSX=XHS(N) : HSY=YHS(N)
BP=BP(N)
Screen Open
1,320,YSIZE*16,2^BP,Lowres
Curs Off : Flash Off
Ink 0 : Bar 0,0 To XSIZE,YSIZE*16
Get Sprite Palette
Screen To Back 1
Paste Icon 0,0,N
Get Bob Length(1)+1,0,0 To XSIZE,YSIZE
Hot Spot Length(1),HSX,HSY
Inc N : Until N=L
F2$=F$
Show On
F$=""
While F$=""
F$=Fsel$("*.ABK",F2$,"Save Sprite bank
as:")
If F$="" Then End
If Upper$(Right$(F$,4))<>".ABK" Then
F$=""
Wend
Save F$,1
```
■

WALKIES

For those of you trying desperately to work out how to make creatures move in a realistic way, here are two examples which should be of use. It is important to match feet between frames in order to convey the illusion of movement realistically.

If you decide to use these animations – or would like to see more, please drop us a line.





This Newsletter was written by Aaron Fothergill, unless otherwise stated, and put together on a Mac by Chris Payne and Visual Eyes.

continued from page 14

put them in your AMOS program (with BOBS on the top!!!)

• A better music extension, with synthetic instruments, to save a lot of memory.

• A better REQUEST extension, with what I think is a fabulous function:

=REQUEST("questions","Cancel","Ok")

... this function will open the requester above all AMOS display, with NO INTERFERENCE AT ALL with your program! It will be a very easy way of displaying alert boxes! It will return 0 or -1 depending on the option chosen.

At the same time, I'm also programming the compiler as the main project. AMOS compiler will be very close to STOS one: It will come as an accessory, and use the same graphic display. You'll have a "setting" menu to change the compiler options.

I want to make this compiler as easy to use as the STOS one: Just put the program in it, watch it work, and that's it. I also want to allow the advanced user to set it to its own desire. As AMOS, it will have two approachs: immediate simple use, and advanced use.

The compiler will output, as STOS, two different types of programs:

- Without the system, to be called within AMOS.
- With AMOS system linked to it, as a standalone program

But I want to add a very special one, designed for un-expanded A500: it will produce a program on the boot block of a disquette. This program will take total control of the machine, starting at address 256! You will win around 100k of memory for your program! (on the bad side, the only I/O function available will be LOAD and SAVE, and you won't have many graphic functions...). But I think this option will be invaluable to professional games writers...

If you have other ideas of new commands, new extensions, please send them to the club (or Mandarin). I also would like you to send all your programs to the club. We have to build a big PD library so that everybody benefit from it. Sprites, musics, procedures, & lots of things can be grabbed from an AMOS program...

If you discover a bug, report it to Mandarin, but please, be sure to:

• Find a way to make this bug happen another time

• Describe all your system: A500, A1000, A2000, A2500,A3000, A8000 (sorry, not last one), memory expansions, hard drives,workbench closed or not etc...

• Describe EXACTLY the way to make this bug happen. Send the faulty program.

All this simple points will help me chasing the bug, and remove it.

Oh yes, talking about bugs: A lot of you have reported bugs in sequential file handling, TYPE MISMATCH errors while inputting. I know my sequential files in V1.1 are not very fast, but they are not bugged! When you do a INPUT #1,A$,B, you MUST be sure there is no comma in the string you are inputting! If there is, INPUT will just take the string up to the comma into A$, and the rest will be affected to B, and will create a type mismatch error.

In order to input string, you should choose LINE INPUT instruction that ignores commas.

I can tell you that sequential files in V1.2 are three times faster.

That's it! Thank you for reading me up to there. Now you can read good Oxford English by Aaaaaaron. Happy AMOSsing!

François Lionet

EXTRAS CORRECTION

There is a bug in the sound tracker converter on version 1 of the Amos Extras disc.

Find the line:

OF_MUSIC = $438

and replace with

OF_MUSIC = $43C

This corrects the timing problem when there are four voices.

Printed by Creative Printers 104 Boutport Street Barnstaple North Devon
Telephone (0271) 77655