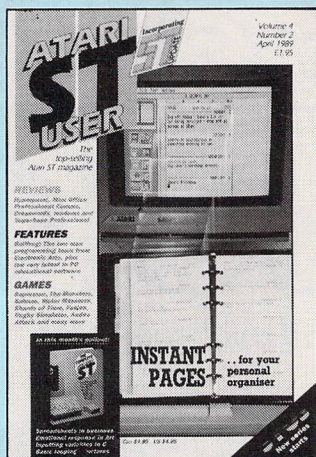




Vol 4 No. 2 April 1989

Managing Editor
Derek Meakin

Group Editor
Alan McLachlan

Features Editor
Roland Waddilove

Production Editor
Peter Glover

Assistant Editor
Nic Outterside

Advertisement Manager
John Snowden

Advertisement Sales
Gail Blicow
Andrea Fawkes

Reviews coordinator
Pam Turnbull

Promotions Editor
Christopher Payne

Promotions Coordinator
Robin Nixon

News Editor
Mike Cowley

Tel: 0625 878888 (All depts)
0625 879940 (Subs)

Telex: 9312188888 DB

Telecom Gold: 72:MAC001

Prestel Mailbox: 614568383.

Published by:
Database Publications Ltd,
Europa House, Adlington Park,
Adlington, Macclesfield SK10 4NP.

ABC

33,706 July-December 1988
ISSN 0952-3006

Subscription rates for
12 issues, post free:

£19 - UK
£27 Europe (incl. Eire)
£42 Overseas (airmail)

Writing for Atari ST User: Articles and programs relating to the Atari ST are welcome. Articles should preferably be typed or computer-printed using double spacing. Unsolicited manuscripts, discs etc, should be accompanied by a self addressed stamped envelope, otherwise their return cannot be guaranteed. Unless agreed otherwise, material is accepted on an all rights basis.

© 1989 Database Publications Ltd. No material may be reproduced in whole or in part without written permission. While every care is taken, the publishers cannot be held legally responsible for any errors in articles or listings.

Atari ST User is an independent publication. Atari Corp (UK) are not responsible for any of the articles they contain or for any of the opinions expressed.

News trade distribution: Europress Sales & Distribution Limited, Unit 1, Burgess Road, Ivyhouse Lane, Hastings, East Sussex TN35 4NR. Tel: 0424 430422.

Database Publications is
a division of Europress Ltd

FEATURES

News

All the latest stories and events from the ever-expanding world of the Atari ST, including the latest Gallup Chart of top-selling games software.

7

Bullfrog

Nic Outterside meets the programming duo responsible for two new and exciting games for the Atari ST.

50

Midi

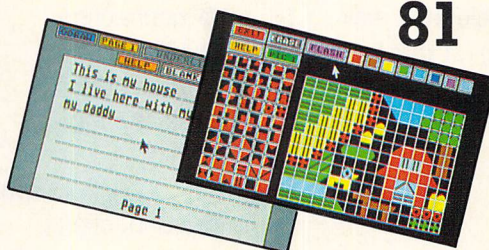
Our music expert takes a trek through the vast range of Midi software and hardware available and gives some pointers on what to buy.

77

Education

We examine a plethora of top quality PD educational software for the ST.

81



STOS

Starting a brand new series, David McLachlan gets to grips with this top selling programming language and passes on some useful hints and tips.

91



Gallery

The combined painting skills of Dave Lowid and Sid Tourtee grace our international art gallery this month.

97

Brillig

Our resident adventure wizard opens the door to give help to intrepid wayfarers everywhere, and includes a review of Sierra Online's Manhunter.

104

MORE FOR YOUR BINDER ...

Part 9 of The Complete Atari ST.
Over the months this illustrated pull-out, which comes free with Atari ST User, is building up into the definitive guide to the working and uses of this powerful micro.

Continuing this month with:

- **Creating mood and emotional response using colour in art.**
- **Some more input functions for programming in C.**
- **Spreadsheets and accounts packages for business use.**
- **We continue looking at Basic's flexible loop structures.**



David McLachlan, one of the STOS design team, starts a monthly feature to help you get the best from this top-selling games creator

Creating multiple miniatures

WELCOME to the first part of a new regular series where together we'll look into and examine the many programming aspects of STOS Game Creator.

STOS provides you with all the commands you need to write your own games. Unlike other Basics, the amateur games writer doesn't need to program his own graphic routines for sprites or animation, as these are provided ready made – I'll cover these later in the series.

To kick off I have written a short utility to illustrate just a few of the wide range of powerful STOS commands. What the routine does is to display up to nine miniature versions of Neochrome or Degas pictures from a disc of artwork.

You could, for instance, use this technique as a novel menu selection device in your own games.

When you have typed in the listing, enter RUN and the program will look for Neochrome picture files from drive A. To use another drive simply alter the drive number on line 40: Drive zero is drive A, drive one is B and so on.

If you want to access picture files in another folder, again it's a simple matter to alter the variable `PATHNAME$` to the desired folder. For instance:

```
PATHNAME$="A:\PICTURES\*.NEO"
```

will access Neochrome files in the folder PICTURES on drive A. To display Degas pictures change the NEO extension in `PATHNAME$` to PII.

When the pictures have been miniaturised and displayed, move the mouse pointer over them to restore their palettes – unless you've used the same one for all your pictures. Click a mouse button to save a copy of the screen. Experiment with the program. Try altering it as suggested. Examine how it's structured and how it works.

Looking at the program, the first unfamiliar command you will come across is KEY OFF. This removes the function key strip from the top of the screen. HIDE ON removes the mouse pointer and CURSE OFF removes the cursor.

Notice that KEY OFF is used before the MODE command. This is because KEY OFF does a clear screen and if MODE was first the function key banner would momentarily reappear then be cleared by KEY OFF, causing it to flicker as the program started. The SCROLL OFF command prevents the screen from

scrolling when text is printed at the bottom.

On the next line you will see the RESERVE command. Among other things this is used to reserve a memory bank – a protected area of ram – for future use. You can reserve memory banks for storing complete screens, and two types of screen are available – permanent and temporary.

When you come to save your program

permanent screen banks will be saved along with it. Temporary screens will be discarded. With the RESERVE command you can also set aside memory banks for data, work areas, character sets and so on. We'll look at some of its other uses in a future article.

When you reserve a bank of memory, no matter what it's going to be used for,

Turn to Page 92 ►



◀ From Page 91

the syntax requires a bank number. When reserving a bank as a screen you can specify any bank number from 1 to 15 provided it isn't already in use, for instance:

RESERVE AS SCREEN 5

In order to load the pictures the program needs to know the contents of the disc, so two commands are provided that allow you to examine each filename. These are DIR FIRST\$ and DIR NEXT\$. The syntax is:

FSP\$=DIR FIRST\$(pathname\$,flag)
FSP\$=DIR NEXT\$

Both commands assign a string, in this case FSP\$, to a 42 character block. DIR FIRST\$ describes the first file in the directory set by PATHNAME\$ and DIR NEXT\$ describes the next. The variable flag is a way of filtering out unwanted types of files from your selection.

The possible file types are as follows:

Bit	Type
0	Normal read/write files
1	Read only files
2	Hidden files
3	Hidden system files
4	Volume labels (the disc title)
5	Folders
6	Files which have been written to and closed

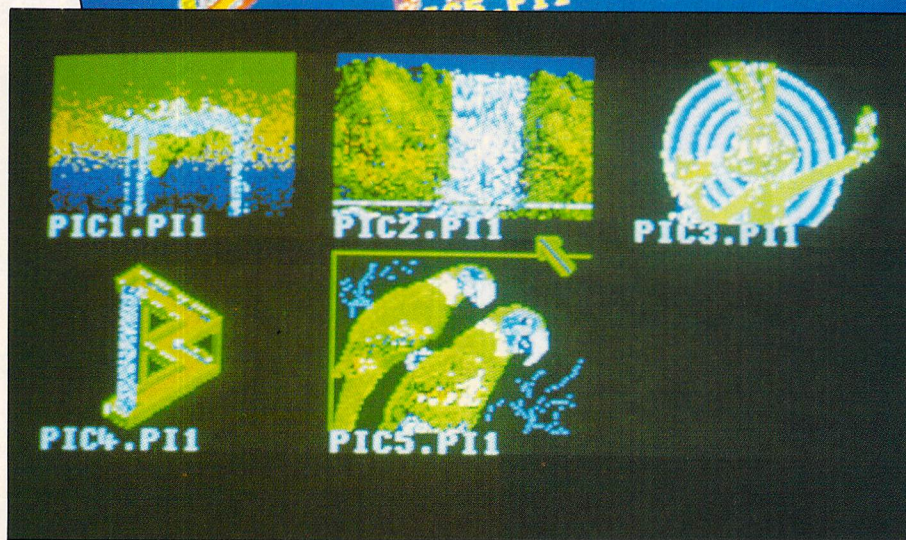
Setting a bit to one will include that file type, for example, if you only want to include normal read/write files (bit zero) and hidden files (bit two) the flag should be set to %0000101 (binary) or 5 (decimal). The path name filters out all unnecessary files, so we don't need to work out which file types we need to exclude. So we set FLAG to -1 to include all files.

The 42 character block that both DIR commands return is constructed as follows:

Character	Usage
0-12	Filename
13-21	Length of the file
22-32	Date the file was saved
33-41	Time the file was saved
42	File type

When you have the character block in the string, you can then print the filename using the LEFT\$ function to extract the first 13 characters. Once the filename is known you can load a Neochrome picture directly into the reserved bank – or, in fact, any other screen.

The program loads the pictures into the



Restore a picture's palette by moving the pointer over it

reserved screen bank five using:

LOAD fsp\$,5

When you specify the bank you are simply giving the command a load address. STOS works this out from the bank number and loads the data into it. You could just as easily load Neochrome files directly to the screen. The address

of this is controlled by three system variables, PHYSIC, BACK and LOGIC. PHYSIC holds the address of the picture currently being displayed on the monitor screen. BACK holds the address of the background screen which is used to update the sprites, like the mouse pointer.

When a sprite is printed STOS has to

Turn to Page 95 ▶

```

1 rem Picture Library
2 rem By David McLachlan
3 rem (c) Atari ST User
10 key off : hide on : curs off : m
ode 0 : scroll off
20 reserve as screen 5
30 dim PALT(9,15)
40 drive=0 : PCNO=1 : PATHNAME$="a:
*.neo" : FLAG=-1
50 FSP$=dir first$(PATHNAME$,FLAG)
60 if FSP$="" then print "No Neochr
ome files available"
70 repeat
80 load FSP$,5
90 gosub 220
100 gosub 300
110 FSP$=dir next$
120 until FSP$="" or PCNO>9
130 screen copy physic to back : sho
w on
140 while mouse key<>2
150 OZ=Z : Z=zone(0)
160 if Z<>OZ and Z<>0 then for C=0 t
o 15 : colour C,PALT(Z,C) : next
170 wend
180 while mouse key<>0 : wend
190 save "a:Pic_dir.neo"
200 end
210 rem *** Reduce and display ***
220 read A,B,C,D,X,Y
230 get palette (5)
240 reduce 5 to physic,A,B,C,D
250 set zone PCNO,A,B to C,D
260 ADD=start(5)+32000 : for C=0 to
15 : PALT(PCNO,C)=deek(ADD) : inc ADD
: inc ADD : next
270 inc PCNO
280 return
290 rem***** Extract file name from
the 42 character parameter block *****
300 pen 15 : locate X,Y : print left
$(FSP$,13);
310 return
320 rem
330 rem***** Data for reduce areas,
zones and filename positions *****
340 data 0,0,96,64,0,7,112,0,208,64,
14,7,224,0,319,64,28,7
350 data 0,68,96,132,0,16,112,68,208
,132,14,16,224,68,319,132,28,16
360 data 0,136,96,199,0,24,112,136,2
08,199,14,24,224,136,319,199,28,24

```


◀ From Page 92

remember what was underneath it so the background can be restored. When a sprite is moved STOS uses the background screen to put back the data that used to be underneath it.

The third screen variable, LOGIC, holds the address of the screen used to draw on when you use a graphics command.

If this isn't the one you are displaying on screen, when you want to display it you use SCREEN SWAP.

You can load a Neochrome file straight into the address PHYSIC or BACK. However, STOS is designed to recognise a NEO or PII file extension so you can load a picture on to the screen without specifying an address. Try loading a Neochrome or Degas picture from the program editor to the background screen using:

```
MODE 0:LOAD "filename",BACK
```

Now move the mouse around. You will see that because the picture is in the background screen STOS updates the mouse pointer with it and the picture is effectively painted on to the monitor screen. Now try loading the picture to the

physical screen as well as the back using:

```
10 MODE 0
20 LOAD "a:filename.NEO",BACK
30 LOAD "a:filename.NEO",PHYSIC
```

When the program has finished loading try moving the mouse pointer around. You will see that the picture doesn't get erased. This is why the utility uses the SCREEN COPY command to copy the physical screen to the background. Its syntax is:

```
SCREEN COPY screen/bankno,x1,y1,x2,
y2 to screen/bankno,x,y
```

This is one of the most useful commands in STOS and is used frequently in writing games.

Screens are 32,000 bytes long. There are also an additional 16 words which are used to hold the 16 numbers of the colour palette. These are stored in an array for future use. When a picture is loaded into bank five the program calls a subroutine to reduce and display miniature pictures and set up zones.

The REDUCE command is a simple one, the syntax being:

```
REDUCE bankno TO bankno,x1,y1,x2,y2
```

Remember, the *bankno* variables are just addresses, so these can be screens too, such as PHYSIC and BACK.

SET ZONE is a simple command. It sets up an imaginary rectangle on the screen so you can detect whether or not a sprite is inside it. The syntax looks like this:

```
SET ZONE zoneno,x1,y1 to x2,y2
```

We use the ZONE command in this instance so we can detect which mini picture we are pointing to with the mouse. The ZONE command:

```
z=ZONE(n)
```

asks what zone sprite number *n* is in. If it returns a zero, the sprite isn't in a zone. For this program we ask what zone sprite zero – the mouse pointer – is in. The number that the ZONE command returns can be used to retrieve the corresponding colour palette stored in the array.

● We have covered quite a few commands here, some of which I will go into more detail later in the series. Next month I'll look at moving, animating, detecting – and that zone command again.

EDUCATIONAL SOFTWARE

SPELLBOOK

Simple to use. Mouse controlled spelling game. Forget the keyboard. Improves hand/eye co-ord. Extremely effective and entertaining educational program. Two age ranges – 4-6 and 7+ **£19.95**

THINGS TO DO WITH WORDS

3 programs to consolidate spelling word and sentence structure. Anagrams:- rearrange the words. Word Hunt:- discover words within words. Jumbled Sentences:- make sense from sentences. Mouse controlled with no keyboard distraction **£19.95**

THINGS TO DO WITH NUMBERS

Maths from a different angle! Time Teller:- Tell the time using the mouse 12 hour and 24 hour clock. All Aboard:- multiplication and division needed to answer these. Book Search:- subtraction and addition handling small numbers. **£19.95**

SOFT STUFF

SPELLING PACKS

4-6 or 7+. Two complete alphabets of words and pictures on 2 disks. Expand the words on offer through 'Spellbook! Gain inspiration to design your own pictures with Alphabet Creator **£14.95**

SPELLBOOK ALPHABET CREATOR

compatible with Degas & Neochrome – allows the creation of unlimited words and pictures – comes with P.D. Neo on disk **£9.95**

Please send me:

SPELLBOOK (4-6) ☐ 7+ ☐
 THINGS TO DO WITH WORDS ☐
 THINGS TO DO WITH NUMBERS ☐
 SPELLBOOK ALPHABET CREATOR ☐
 SPELLING PACKS (4-6) ☐ 7+ ☐

NAME _____
 ADDRESS _____

Please make cheques and PO's payable to Softstuff and send to **Softstuff, 193 Sketchley Road Burbage Leicestershire LE10 2DY Tel: 0455 617426**

Softstuff, 193 Sketchley Road Burbage Leicestershire LE10 2DY

Bombs away!



In Part 2 of his STOS series David McLachlan shows you how to swap screens and detect those sprite collisions

I HAVE written a small demonstration program this month which contains some common techniques used to write games. They aren't just restricted to STOS though. Once you have an understanding of how they work you should be able to adapt them to any programming language, although it won't be as easy a task as with STOS.

The commands covered in this article include SCREEN SWAP – vital for smooth graphics and animation – and a full explanation of the tricky COLLIDE function.

As we will be using sprites you will need to design some yourself, as they aren't included in the listing. Consult the table opposite which describes the various sprites used in the game and their positions.

The program repeatedly moves a laser

base and spaceship from left to right then back again, each at a different speed. To spice the action up a little, asteroids at the top of the screen start to move at random. The idea of this game is to fire missiles from the laser base to hit either a moving asteroid or the spaceship.

At first you might think the program looks rather complicated, but once broken down into its various routines it's quite simple. Its structure is as follows:

- Set up variables and arrays.
- Start of loop.
- Print stars, asteroids and sprites.
- Screen swap.
- Check for collisions.
- Goto loop start.

The first thing the program does is to call a subroutine that sets up screen five along with various variables and zones. The powerful command SCREEN\$ acts like a software blitter and with it you can copy sections of the screen into a string, or vice versa. The syntax is:

```
a$=SCREEN$(screen,x1,y1 to x2,y2)
SCREEN$(screen,x1,y1)=a$
```

The first line shows how a rectangular area of the screen can be stored in the string *a\$*. The parameter *screen* is either a screen or a bank number and *x1,y1,x2,y2* are the coordinates of the screen rectangle to store.

In the second example the syntax alters slightly, and it now allows you to



copy an area of the screen stored in the string *a\$* to the screen or bank number *screen* at the coordinates *x1,y1*.

In the game the asteroid sprite is copied using SCREEN\$ into the string *asteroid\$*. "You've spelt asteroid wrong", you're probably thinking. Well you can't use it as a variable because the word contains the letters AS – a STOS command. STOS will separate AS from *asteroid* thinking it's a command followed by the variable *teroid*. Try it and see. That's also why the variable for the laser is spelt *lazer*.

Once we have a copy of the asteroid in *asteroid\$* the screen is cleared and the stars are printed. In order to demonstrate screen swapping the game doesn't use sprites for the asteroids. A screen full of stars is plotted and copied into bank five. We will have to reprint them later in the program, otherwise the asteroids will rub them out when they move.

We now print the rest of the asteroids – seven in all – across the top of the screen at random heights. As this is done, zones are set up around the asteroids so later we can check for collisions. Also we place small invisible sprites where the asteroids are, assign a movement pattern to them and switch the movement on. I'll explain the reason and how to do this a little later.

We now print the rest of the sprites, then it's head first into the main loop, where all the action takes place.

Screen swapping

First of all let me explain about screen swapping. The idea is to print all the updated graphics on a screen stored in memory – not the one displayed on the monitor – then swap this with the screen you can see. STOS Basic has a clever screen variable called LOGIC which allows you to swap the physical screen – controlled by the screen variable PHYSIC, with the logical screen – con-

trolled by the screen variable LOGIC. You can see a full explanation of these screen variables in last month's article.

At first LOGIC is set to the same address as PHYSIC so screen swapping won't make any difference, however it will if LOGIC points to a different screen. In our program we point LOGIC at the background screen, the address of which is held by BACK.

When we want to print anything on the screen we must print it to the LOGIC screen, which now can't be seen because it is in the background. Then we use SCREEN SWAP to exchange the LOGIC screen with the PHYSIC screen.

The graphics which you have printed are now displayed and you can print updated graphics, like a sprite, in a different position on the now vacant LOGIC screen. This technique is used in practically all commercial games to produce flicker-free sprites.

Printing graphics

The first task performed in the main loop is to call another subroutine to handle the printing of all graphics. It is carried out in the following order: First the star background is copied from bank five on to the logical screen, deleting anything that used to be there, then the asteroids are printed and finally the sprites are displayed.

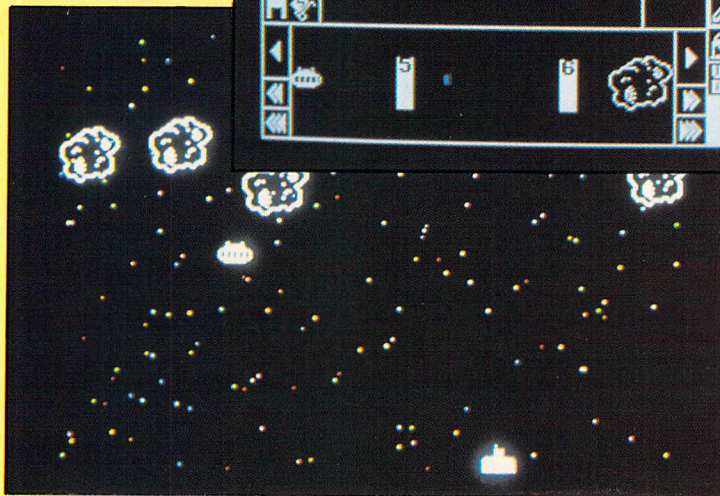
The asteroids are controlled in a similar way to the bouncing Orbit logo in one of the three free games provided with STOS. This is a very useful way of controlling the path of large sprites without having lots of IF statements to change their direction. I'll explain this shortly, but first, an explanation of the sprite command is in order. It is really quite simple to use. The syntax looks like this:

```
SPRITE s,x,y,i
```

It displays sprite *s* at the coordinate *x,y* using the image *i*. You are allowed up to 15 sprites so *s* can be any number between one and 15. The images are stored in sprite bank one and you can have any number of images, memory permitting of course. They can be created easily using the sprite editor.

When you define the sprites required for this game use the Quit & Grab option

Creating the sprites using the editor



The finished game

in the file section of the sprite editor. This grabs the sprites from the editor and places them into bank one.

Typing LISTBANK will list the banks that are defined. You should find sprite bank one will be listed. The X coordinate of a sprite can range from -640 to +1280, which means it can come and go off the side of the screen. The same can be done with the Y coordinate, but the range is only -400 to +800.

Moving sprites

The MOVE command allows you to define paths for sprites to follow. There are two such commands, both performing the same function except on different axes:

```
MOVE X s,"start(speed,step,count)l end"
MOVE Y s,"start(speed,step,count)l end"
```

Both commands are more or less the same, so I will only explain the MOVE Y

command. The variable *s* is the sprite you wish to move. The next parameters allow you precise control over the sprite. The number *start* is the Y location you wish the sprite to start from - an optional parameter. If you omit it the sprite starts from where you placed it on the screen.

Next *speed* defines the delay in fiftieths of a second between each movement of the sprite. The *step* is how many pixels you wish the sprite to move with each time, and *count* is how many times you would like it to perform this function. This is also optional, and setting it to zero will have the effect of *count* being infinite.

The last parameter, *end*, is also optional and will specify an end coordinate of the sprite's movement provided that it can be at that coordinate.

If the step takes you past the end coordinate you will find that the sprite won't stop until *count* has finished. You can also have an optional *l* just before the *end* and this will make the movement loop back to the start. The options in brackets can be repeated to string together several movements.

The program's movement command looks like:

```
MOVE Y sp,"(3,1,10)(3,-1,10)l"
```

In conjunction with the other MOVE command the sprite can be made to whizz around the screen along a complicated path. To set it going you use the

Turn to Page 100 ►

Image number	Description	Size	Hot spot coordinate
1	Laser base	16 x 11	0,0
2	Spaceship 1st frame	16 x 9	7,4
3	Spaceship 2nd frame	16 x 9	7,4
4	Spaceship 3rd frame	16 x 9	7,4
5	Bullet	16 x 4	0,0
6	Asteroid	32 x 25	0,0

Use the editor provided with STOS to create six sprites

◀ From Page 99

command MOVE ON *s*, where *s* is the sprite you wish to start moving. Issuing this without specifying a sprite number will result in all the sprites you have defined with movement to start moving.

The position of the asteroids are now printed where their respective invisible sprites are. The sprites' Y position is found by using the command Y SPRITE:

```
y=Y SPRITE(s)
```

The variable *s* is the number of the sprite whose Y coordinate you want to find and *y* is the value returned. You can ask the X position of a sprite with the similar command:

```
x=X SPRITE(s)
```

So when one of the little invisible sprites moves up and down we can find its Y position and then use SCREEN\$ to put an image of the asteroid at the sprite's Y location.

Now we print the spaceship and laser base then swap screens. We don't define movement of the laser base and spaceship because this is on interrupt. Unfortunately the interrupt isn't in synchronisation with our screen swapping, so if we were to use the movement command the sprites would appear to be flickering.

The spaceship's direction is controlled by the variable *sd*, which is set to -2. In line 920 we subtract -2 from the X position of the spaceship. The result of this is to add two because two minuses make a plus. When the spaceship reaches the

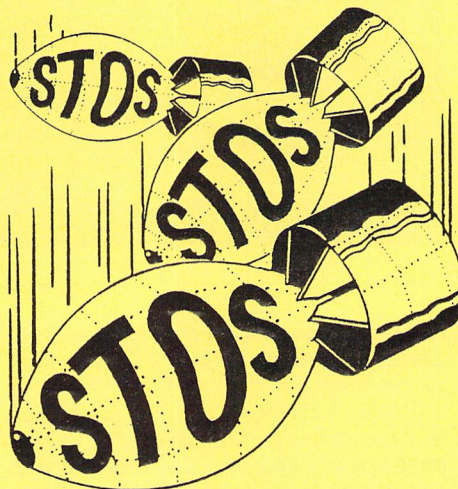
side of the screen its direction is reversed. We make *sd* equal to -*SD*.

Crash course

Now for collision detection. If you have played around with the COLLIDE function you might have found it a bit confusing. I'll explain exactly how it works. The syntax of the command looks like this:

```
n=COLLIDE(s,w,h)
```

The variable *s* denotes which sprite you want to check and *w* and *h* are the variables to set the width and height of an



imaginary box around the sprite. Contrary to what the manual tells you, the hot spot is, in fact, in the centre of this box.

Whenever STOS draws a sprite you specify its coordinates in terms of this point on the screen. As a default the hot

spot is always at the top left of a sprite but you can change this using the sprite editor. When you define the spaceship you will have to set its hot spot in the centre.

The value that the collide command returns is a bit mapped number indicating which sprites, sprite *s* has collided with. Each bit of the two byte number represents a sprite. Bit zero is the mouse pointer, as this is sprite zero, bit one represents sprite one, bit two is sprite two and so on up to 15. If, for example, the collide command returned the number six the bit map would look like this:

```
0000000000000110
```

which means that sprite *s* would have collided with sprites one and two.

All that remains is to use INKEY\$ to check the spacebar for firing the bullet. When you press a key it is inserted into a buffer and INKEY\$ gets the next character out of it. We make *k\$* equal to INKEY\$ and check if it is a space. If it is, we set a flag denoting that firing is in progress and we make a shooting sound. We now can check for the fire flag and move the bullet if and when the check proves positive.

When a collision is detected between the spaceship and the bullet, we set the fire flag back to its original state, and to add a bit of atmosphere use a BOOM to add sound. We also do the same when we detect the bullet sprite entering the zone of a moving asteroid.

● Next month I'll create a simple bat and ball game to demonstrate the Play command and changing the mouse pointer.

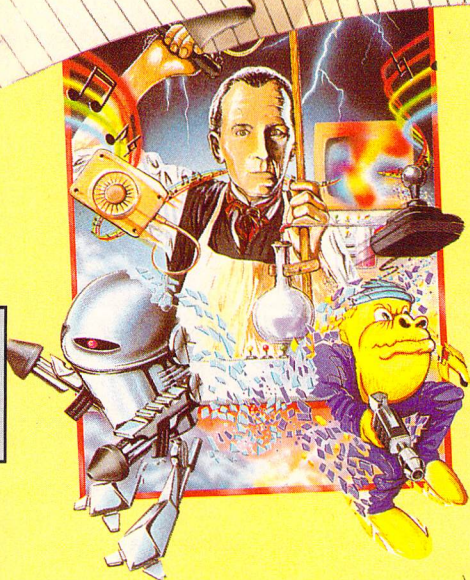
```
1 REM Zap 'ep By D. McLachlan
2 REM (c) Atari ST User
10 KEY OFF : MODE 0 : CURS OFF : HI
DE ON : CLICK OFF
20 GOSUB 2000
30 LOGIC=BACK : MOVE ON sp
40 REM***** MAIN LOOP *****
50 WHILE-1
60 GOSUB 1000
70 IF Y SPRITE(3)<0 THEN fyre=0
80 k$=INKEY$
90 IF k$=" " AND fyre=0 THEN SPRITE
3,X SPRITE(1)+8,Y SPRITE(1),bullet :
fyre=1 : SHOOT
100 IF COLLIDE(3,8,4)=4 THEN SPRITE
OFF 3 : fyre=0 : BOOM
110 n=RND(50) : IF n=1 THEN MOVE OFF
sp : sp=RND(6)+4 : MOVE Y sp,"(3,1,1
0)(3,-1,10)" : MOVE ON sp
120 IF sd=-2 THEN DEC spaceship : IF
spaceship<2 THEN spaceship=4
130 IF sd=2 THEN INC spaceship : IF
spaceship>4 THEN spaceship=2
140 IF X SPRITE(1)<2 OR X SPRITE(1)>
302 THEN d=-d
```

```
150 IF X SPRITE(2)<-64 OR X SPRITE(2
)>332 THEN sd=-sd
160 z=ZONE(3) : IF z=sp-3 THEN SPRIT
E OFF 3 : fyre=0 : BOOM
170 WEND
999 REM***** Print Graphics *****
1000 SCREEN COPY 5 TO LOGIC
1010 FOR x=0 TO 6
1020 SCREEN$(LOGIC,x*48,Y SPRITE(x+4
))=asteroid$
1030 NEXT x
1040 IF fyre=1 THEN SPRITE 3,X SPRIT
E(3),Y SPRITE(3)-8,bullet
1050 SPRITE 2,X SPRITE(2)-sd,100,spa
ceship
1060 SPRITE 1,X SPRITE(1)-d,188,laze
r
1070 UPDATE : SCREEN SWAP : WAIT VBL
1080 RETURN
1999 REM ***** Set up *****
2000 RESERVE AS SCREEN 5
2010 DIM y(7) : sd=-2 : d=-4 : sp=7
: spaceship=2 : bullet=5 : lazer=1 :
sc=0
```

```
2020 SPRITE 1,1,y(1),6 : UPDATE : PU
T SPRITE 1 : SPRITE OFF : WAIT VBL
2030 SET ZONE 1,0,y(1) TO 32,y(1)+25
2040 asteroid$=SCREEN$(PHYSIC,1,y(1)
TO 32,y(1)+25) : CLS
2050 FOR x=1 TO 7 : y(x)=RND(5)*10 :
NEXT
2060 FOR s=1 TO 200 : INK RND(15) :
PLOT RND(319),RND(199) : NEXT
2070 SCREEN COPY PHYSIC TO 5
2080 FOR x=0 TO 6
2090 SCREEN$(PHYSIC,x*48,y(x+1))=azt
eroid$
2100 SET ZONE x+1,x*48,y(x+1) TO (x*
48)+31,y(x+1)+25
2110 SPRITE 4+x,x*48,y(x+1),7 : MOVE
Y 4+x,"(3,1,10)(3,-1,10)"
2120 NEXT
2130 ANIM 2,"(2,5)(3,5)(4,5)" : MOV
E X 2,"-166(1,1,502)(1,-1,502)"
2140 MOVE X 1,"0(1,2,152)(1,-2,152)"
2150 SPRITE 2,-16,100,2 : SPRITE 1,0
,188,1
2160 RETURN
```




David McLachlan makes music with STOS and shows how to write a bat and ball game



Play it again Sam

WELCOME once again to our regular STOS programming feature where we take a look at various aspects of this game creation package. This month we are going to examine the PLAY command and see how to use the mouse pointer as a sprite. To this end I've written a small bat'n'ball game – nothing too complex, just a simple squash session.

It features a ball bouncing around an area enclosed on three sides. Using the mouse to control a bat you have to stop the ball flying off the bottom of the screen. To add a little variety there's a magnet in the centre of the screen. If the ball gets too close it will be deflected and fly off in another direction.

The program uses the PLAY command to make one of two noises depending on whether the ball hits a wall or the bat. The syntax for the command is:

PLAY v,p,d

where *v* is a voice from one to three, *p* is the pitch – any number between 0 and 96 and *d* is the duration in 50ths of a second.

Contrary to what you might expect, what this command actually does is to play the sound for longer than the duration. However, if another sound is executed that is played immediately, otherwise the original sound keeps on going.

Type the following at the OK prompt in STOS:

**VOLUME 1,15
PLAY 1,37,50**

This will produce a middle C on voice 1 at volume 15. The program will wait for a second before you can assign another pitch to that voice.

Hand in hand with the PLAY command is VOLUME. The syntax for this is:

VOLUME v,i

where *v* is the voice number between one and three and *i* is the intensity. Before you set this however, use CLICK OFF to turn the key clicking off because this interferes with the PLAY command. CLICK ON re-activates key clicking.

The note played in the last example but

one won't stop because it uses a continuous pure tone. If you want to stop it you have to play another note. Either play one with a pitch of zero or play the original note through an envelope.

An envelope is used to alter the volume of a sound being played. It determines whether the sound gets progressively louder or quieter, repeatedly gets louder and so on. There are 15 types of envelopes available using the command:

ENVEL t,s

The variable *t* is the type of envelope and ranges from 1 to 15 and the second parameter *s* defines the speed at which the envelope affects the sound. To use it in conjunction with the PLAY command the voice you are going to use must have its volume set at 16. Type in and notice the difference between these two lines:

**VOLUME 1,16:ENVEL 10,500:PLAY 1,37,1
VOLUME 1,16:ENVEL 10,5000:PLAY 1,37,1**

Try increasing the speed even more and you will be surprised by the effect produced.

In the game listing this month I use envelope pattern two which starts loud and then trails off to nothing.

The NOISE command produces white noise which is like the hiss you can hear on the television when the station goes off-air. Its syntax is:

NOISE v,p

The first parameter *v* is the voice and *p*

SPRITE INFORMATION FOR USE WITH SQUASH LISTING

You'll need to define six sprites using the sprite editor before you can play this game. The following table describes each sprite's use, its size in pixels and where the hot spot is. These should be saved as a single sprite file and loaded with the Basic program:

Sprite number	Sprite description	Sprite size	Hot spot coordinate
1	Bat	48 x 10	23,4
2	Ball	16 x 5	2,2
3	Magnet	16 x 16	7,7
4	Brick wall 1	16 x 16	0,0
5	Brick wall 2	16 x 16	0,0
6	Brick wall 3	16 x 16	0,0

Turn to Page 64 ►

◀ From Page 63

is the pitch, ranging from 1 to 31. You can use this just like the PLAY command. Try it in the last example.

Sprites

STOS claims to be able to display 15 sprites at once. It can in fact, display 16 as the mouse pointer is a sprite too. Although it cannot be controlled by the MOVE, ANIM and SPRITE functions, it can be manipulated and made to act like a sprite using the various mouse commands.

I pointed out in the first article in the April 1989 issue that the mouse pointer is sprite zero.

STOS has three different predefined images for the mouse pointer and you can select any of these using the CHANGE MOUSE command:

```
CHANGE MOUSE n
```

The number *n* is the image. However, you aren't limited to just these three predefined images. You can change the pointer to any image in the sprite bank.

The type of game written here is much easier to play with a mouse than with a joystick, so the pointer is defined as being the first image in the sprite bank – the bat.

To prevent the pointer – now the bat – from wandering all over the screen LIMIT MOUSE will restrict the pointer to a rectangular area. The syntax looks like this:

```
LIMIT MOUSE x1,y1 TO x2,y2
```

The first two coordinates *x1,y1* are the top left and *x2,y2* are the bottom right coordinates of the rectangular area. In the game this command restricts the mouse pointer to a one pixel high strip along the bottom of the screen.

Other commands relating to the mouse pointer include XMOUSE and YMOUSE. These allow you to read the position of the mouse. For example:

```
x=XMOUSE  
y=YMOUSE
```

The mouse pointer's position can also be set using the same commands but with a slightly different syntax:

```
XMOUSE=100  
YMOUSE=100
```

This will position the mouse pointer at the coordinate 100,100.

The game listing nests three WHILE ... WEND loops with the bulk of the program between them. The innermost loop is active when the ball is in play.

The ball is controlled by two variables

using Boolean logic. The ball is kept within the boundaries of the screen by reversing either its X or Y direction, depending whether it reaches the sides or the top of the screen. Its direction is also altered when it collides with the bat.

Collisions are detected using the COLLIDE function which was explained in detail last month. When the ball hits something a subroutine is called which uses the XMOUSE command to check where on the bat it has hit. The result of these checks determine at what angle the ball bounces off.

The method used to move the ball is neat and worth remembering. The sprite is put on the screen using its own coordinates plus or minus its direction, depending on what direction the ball is moving.

There's another sprite in the centre of the screen – the magnet. A zone is set up around the sprite and using the ZONE function we test to see whether the ball is near the magnet. A flag is set to indicate true or false. If true then the program calls a subroutine to deflect the ball in a random direction.

An extra flag is set because although the ball might have been sent on another path it still might be within the magnet's zone and we don't want the subroutine called again until the ball has left it.

● *Next month a look at using pull-down menus and how to incorporate them into a simple Simon game.*

```
10 REM Squash
20 REM By David McLachlan
30 REM (c) Atari St User
40 REM Use the mouse!
50 KEY OFF : CURS OFF : MODE 0
60 PALETTE $0,$511,$226,$337,$447,$3
33,$555,$777,$700,$600,$400,$222,$115
70 FOR L=0 TO 19 : SPRITE 1,L*16+1,0
,5 : UPDATE : PUT SPRITE 1 : NEXT
80 PEN 7
90 again$=""
100 WHILE again$=""
110 VOLUME 1,16 : VOLUME 2,16 : VOLUM
E 3,16 : ENVEL 2,3000
120 CHANGE MOUSE 4
130 LIMIT MOUSE 23,190 TO 295,190
140 SET ZONE 1,144,84 TO 176,116
150 Lives=3 : sc=0
160 FOR L=0 TO 19 : SPRITE 1,L*16+1,0
,5 : UPDATE : PUT SPRITE 1 : NEXT
170 FOR L=1 TO 11 : SPRITE 1,1,L*16,4
: UPDATE : PUT SPRITE 1 : SPRITE 1,30
4,L*16,6 : UPDATE : PUT SPRITE 1 : NEX
T
180 REMbar 0,0 to 12,199 : bar 0,0 to
319,12 : bar 306,0 to 319,199
190 LOCATE 5,0 : PRINT "Score:";sc :
LOCATE 27,0 : PRINT "Lives:";Lives
200 SPRITE 2,160,100,3
210 WHILE Lives>0
220 xd=2 : yd=-2
230 SPRITE 1,240,100,2 : UPDATE
```

```
240 LOCATE 0,9 : CENTRE "Get Ready "
: WAIT 25
250 LOCATE 0,9 : CENTRE "Get Steady"
: WAIT 25
260 LOCATE 19,9 : CENTRE " GO "
: WAIT 25
270 LOCATE 0,9 : CENTRE " "
280 TIMER=0
290 WHILE Y SPRITE(1)<200
300 GOSUB 640
310 c1=COLLIDE(0,24,5)
320 c2=ZONE(1) : IF c2=0 THEN f=0
330 IF c1=2 THEN yd=-yd : GOSUB 530
340 IF c2=1 THEN GOSUB 480
350 IF X SPRITE(1)<16 OR X SPRITE(1)>
304 THEN xd=-xd : PLAY 37,2
360 IF Y SPRITE(1)<16 THEN yd=-yd : P
LAY 37,2
370 IF TIMER>1000 THEN IF yd<0 THEN D
EC yd : TIMER=0 ELSE INC yd : TIMER=0
380 WEND
390 DEC Lives : LOCATE 27,0 : PRINT "
Lives:";Lives
400 WEND
410 CLS : LOCATE 0,9 : CENTRE "Game 0
ver"
420 sc$="Final Score"+STR$(sc) : LOCA
TE 0,11 : CENTRE sc$
430 LOCATE 0,13 : CENTRE "Play again
(Y/N)?"
440 again$="" : WHILE again$="" : aga
in$=UPPER$(INKEY$) : WEND
```

```
450 CLS
460 WEND
470 END
480 ry=RND(2) : f=1
490 IF xd<0 AND f=1 THEN DEC xd
500 IF xd>0 AND f=1 THEN INC xd
510 IF ry=1 AND f=1 THEN yd=-yd
520 RETURN
530 PLAY 1,37,0 : PLAY 2,41,0 : PLAY
3,34,1
540 sc=sc+10*ABS(yd) : LOCATE 5,0 : P
RINT "Score:";sc
550 IF X SPRITE(1)<X MOUSE-16 AND xd<
0 THEN xd=4 : GOSUB 640 : RETURN
560 IF X SPRITE(1)<X MOUSE-16 AND xd>
0 THEN xd=3 : GOSUB 640 : RETURN
570 IF X SPRITE(1)>X MOUSE+16 AND xd>
0 THEN xd=-4 : GOSUB 640 : RETURN
580 IF X SPRITE(1)>X MOUSE+16 AND xd<
0 THEN xd=-3 : GOSUB 640 : RETURN
590 IF X SPRITE(1)<X MOUSE-8 AND xd<0
THEN xd=-2 : GOSUB 640 : RETURN
600 IF X SPRITE(1)<X MOUSE-8 AND xd>0
THEN xd=2 : GOSUB 640 : RETURN
610 IF X SPRITE(1)>X MOUSE+8 AND xd>0
THEN xd=2 : GOSUB 640 : RETURN
620 IF X SPRITE(1)>X MOUSE+8 AND xd<0
THEN xd=-2 : GOSUB 640 : RETURN
630 RETURN
640 SPRITE 1,X SPRITE(1)-xd,Y SPRITE(
1)-yd,2 : UPDATE : WAIT VBL
650 RETURN
```




Continuing his series on STOS programming, David McLachlan shows how to create your own menu bar like Gem

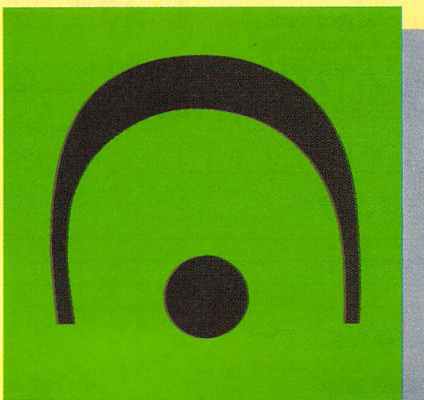
IF you want to write games, but haven't the knowledge or time to program in machine code, STOS will supply you with ready made routines to manipulate animated sprites, music and menus just like professional software.

This series complements the manual by giving more information and longer, more complex example listings. This month I shall examine the MENU command and have written a short game, Simon, to demonstrate how to use the relevant instructions.

You are quite likely to have seen Simon Says before, but for those who haven't the idea is to do as Simon does. There are four different coloured squares on the screen, each having its own musical note. Simon will light up the squares in a particular sequence, playing the notes at the same time. You have to remember which squares Simon lit up and in what order and repeat his sequence.

He starts with one note, and if you get that right adds another to the sequence and so on. You lose a life if you get the sequence wrong. Lose three lives and the game is over.

You input your sequence using the



numeric keypad keys 7,9,1 and 3 which are arranged in a square corresponding to the squares on the screen.

Menu commands

The program uses the MENU commands to set up three pull-down menus, Info, Speed and Start At. On the Info menu there is About Simon and Quit. The former tells you who wrote the program while the latter quits the game.

The Speed menu allows you to choose one of three playing speeds - slow, medium and fast. Use the Start At menu to set the number of notes in the sequence that Simon starts with. The three choices, one, four and eight notes, provide a selectable difficulty level.

Unlike other versions of Basic, setting up menus with STOS is very easy. It stores the items in a special array called MENU\$. The title is defined as:

```
MENU$(1)=" Info "
```

Placing options on the menu is just as easy. Again the MENU\$ array is used, but in a different way:

```
MENU$(1,1)=" Credits "
```

This would put the option Credits in the first position in the first pull-down menu. To activate it you use the MENU ON command, its syntax is:

```
MENU ON border,mode
```

With this command you can choose the



type of border, which range from 1 to 16, the menus will use. There are also two types of menus, Drop Down where they drop down when the mouse pointer touches the menu bar, and Pull Down where you have to click with the mouse button on the menu bar.

Let's take a look at how these commands have been implemented within the example game listing. Instead of individually naming each menu item with a separate menu command, time and space is saved by putting the relevant information in data statements. These are

What's on

```
10 REM Simon - By D. McLachlan
20 REM (c) Atari ST User
30 KEY OFF:CURS OFF:CLICK OFF:MODE 0
40 PALETTE $0,$777,$400,$40,$4,$404
50 ERASE 5:RESERVE AS DATA 5,255:add
=START(5)
60 DIM bx(4,4),col(4,2):speed=20:str
t=1
70 RESTORE 820
80 FOR x=1 TO 3
90 READ n,heading$:MENU$(x)=heading$
100 FOR y=1 TO n
110 READ option$:MENU$(x,y)=option$
120 NEXT y
130 NEXT x
140 .FOR l=1 TO 4
150 READ bx(l,1),bx(l,2),bx(l,3),bx(l,4)
160 NEXT l
170 FOR x=1 TO 4:READ col(x,0),col(x,1),col(x,2):NEXT x
180 MENU$(1,2) OFF:MENU$(2,4) OFF
190 MENU ON 1:CURS OFF
200 ON MENU GOTO 650,730,780
210 ON MENU ON
220 WHILE-1:WEND
230 END
```

```
240 MENU OFF:MODE 0:SCROLL OFF
250 FOR l=1 TO 4
260 INK l+1:BAR bx(l,1),bx(l,2) TO bx(l,3),bx(l,4)
270 NEXT l
280 REM***** Main loop *****
290 lives=3:round=strt
300 WHILE lives>0
310 add=START(5)
320 LOCATE 5,1:PRINT "Round:";round:L
OCATE 27,1:PRINT "Lives:";lives
330 FOR l=1 TO round
340 n=RND(3)+1:n=col(n,0)
350 COLOUR n+1,col(n,2)
360 VOLUME 1,15:PLAY 1,n,speed:PLAY 1,0,5
370 COLOUR n+1,col(n,1)
380 DOKE add,n:INC add:INC add
390 NEXT
400 add=START(5):c=0
410 REM***** Your turn *****
420 REPEAT
430 WHILE INKEY$<>"":WEND
440 LOCATE 13,24:PRINT "Note number:";c+1
450 k$="":WHILE INSTR("7913",k$)=0:k$=INKEY$:WEND
460 k=INSTR("7913",k$)
```


read into the MENU\$ array with a nested FOR ... NEXT loop.

You can switch off various options within the menu using:

```
MENU$(title,option) OFF
```

This shades the relevant menu item, so you know it has been switched off. The command:

```
MENU$(title,option) ON
```

will reactivate the item. I use this command to switch off the dividing lines in the menus.

The coordinates of the squares, plus their colours and tones, are read into arrays. The menus are activated with the MENU ON command, giving them a thin border. Next the ON MENU command is used which allows you to jump to one of a list of line numbers. In Simon the command looks like this:

```
ON MENU GOTO 650,730,780
```

This acts rather like an ON *n* GOSUB command where *n* is a numeric variable. The menus are numbered from left to right, starting at one, so if Speed is selected – menu number two – the program will jump to the second line number in the list.

After the menus are set up the program hangs around line 220 in an infinite loop. ON MENU ON is used to set this command into action. The line numbers in this example are the subroutines which handle the selecting of the menu items.

MNBAR and MNSELECT are two

system variables that hold the menu's status. MNBAR represents the MENU title you have chosen, so you could use ON MNBAR instead of ON MENU.

MNSELECT denotes the number of the menu item that you have selected. This is used to determine where to jump to when an option is clicked on. Line 650:

```
650 ON MNSELECT GOTO 660,0,710
```

tells the program where to go when items on the Info menu are selected. As we have immobilised the second item – the row of dashes – the program will not jump to the second line number, so it doesn't matter what it is.

On line 660 we freeze the menu so it can't be activated when the credits are being displayed. This is the command:

```
MENU FREEZE
```

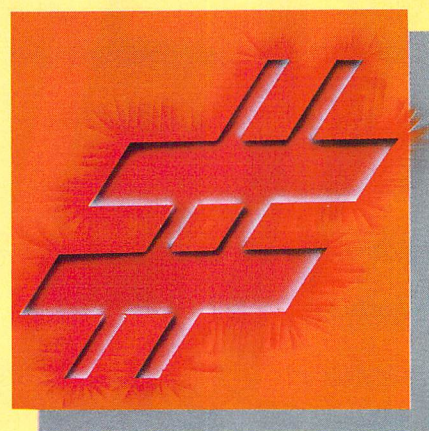
It can be reactivated using MENU ON.

Speed is controlled by the variable *speed* which is used in the PLAY command to set the length of the note. The variable *round* sets the number of notes played in Simon's sequence. The sequence is played in a FOR ... NEXT loop and the next note is selected at random. The pitch is stored in the *col* array as is the colour of the square to be lit. The square is highlighted using the command:

```
COLOUR c,$nnn
```

where *c* is the colour number and *\$nnn* is the colour.

Once the note has been played the



colour is changed back and the square number is DOKED into memory bank five so we can check it against your replay of the sequence.

Once this process has been repeated *round* times the program drops through to where you input your response. The number of the correct square is retrieved by DEEKing bank five and is compared against your guess. If it is different the variable *lives* is decremented.

Icons on the menu

The menus don't necessarily have to be composed of text. You can also use icons. Try loading the STOS icon editor and defining three icons to represent the three Simon menu options, or load the icon file provided.

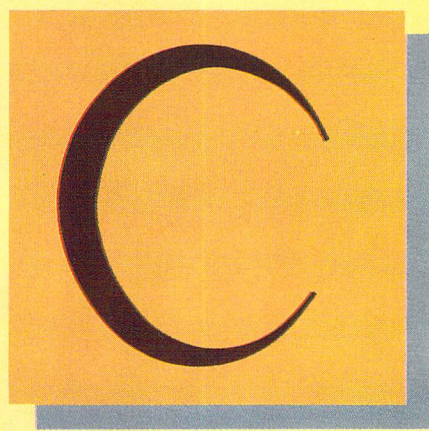
To use them in your menu change line 90 to read:

```
90 READ n,h:MENUS(x)=ICONS(h)
```

Also remove the menu titles in the data statements and replace them with the numbers 26, 43 and 25. Run the program and look at the result.

Now you know how to use the MENU command, why not try to add a few more menus to the game? For example, you could have a choice of shapes to use instead of squares. Or how about two players?

● Next time we'll look more deeply into the screen copy command in a fruit machine simulation.



the menu?

```
470 n=DEEK(add):INC add:INC add
480 COLOUR k+1,col(k,2)
490 PLAY 1,col(k,0),speed:PLAY 1,0,0
500 COLOUR k+1,col(k,1)
510 IF k=n THEN INC c
520 UNTIL c=round OR k<>n
530 LOCATE 26,24:PRINT " "
540 IF k<>n THEN GOSUB 590 ELSE WAIT
50
550 INC round
560 WEND
570 GOTO 70
580 REM***** Oooops wrong *****
590 LOCATE 14,2:PRINT "Oooops Wrong!"
:DEC round
600 PLAY 1,11,20:PLAY 1,0,5:PLAY 1,8,
25:PLAY 1,0,0
610 WAIT 100:LOCATE 14,2:PRINT " "
"
620 DEC lives
630 RETURN
640 REM***** Info *****
650 ON MNSELECT GOTO 660,0,710
660 MENU FREEZE:WINDOPEN 1,8,5,24,12,
1:CURS OFF
670 LOCATE 0,3:CENTRE "SIMON":PRINT:P
RINT:CENTRE "By David McLachan"
680 WHILE MOUSE KEY<>0:WEND:WHILE MOU
```

```
SE KEY=0 AND INKEY$="" :WEND
690 WINDEL 1
700 MENU ON 1:GOTO 200
710 MENU OFF:MODE 1:END
720 REM ***** Set speed *****
730 ON MNSELECT GOTO 740,750,760,0,24
0
740 speed=20:GOTO 200
750 speed=15:GOTO 200
760 speed=10:GOTO 200
770 REM ***** Set number of notes *
*****
780 ON MNSELECT GOTO 790,800,810
790 strt=1:GOTO 200
800 strt=4:GOTO 200
810 strt=8:GOTO 200
820 DATA 3," Info "," About Simon ","
-----"," Quit "
830 DATA 5," Speed "," Slow "," Mediu
m "," Fast ","-----"," Start "
840 DATA 3," Start at.. "," 1 Note ",
" 4 Notes "," 8 Notes "
850 DATA 50,30,150,100,170,30,270,100
,50,110,150,180,170,110,270,180
860 DATA 37,$400,$700,39,$40,$70,41,$
4,$7,42,$404,$707
```




I HAVE written a fruit machine game to show how to manipulate large numbers of sprites for my latest article in this STOS programming series. There's also a very useful tip to find which frame a particular animation sequence is on.

The fruit machine is completely mouse controlled. Clicking on the flashing Start button will set the machine's wheels in motion. If the fruits stop in a winning combination you can either gamble or collect your winnings by clicking on the appropriate button.

As in most games it consists basically of

one main loop with all the subroutines nested inside. If you strip it to its skeleton, in pseudocode it looks like this:

```
Set up
Start of Loop
  Are nudges available?
  Are holds available?
  Wait for start
  Print money
  Spin the wheels
  Check for winnings
  Gamble?
  Print winnings
End of Loop
```

The program first calls a subroutine to unpack and print the background, lay out the screen and initialise arrays, variables and screen strings. Data for the sequence of fruits on each wheel is read into a two dimensional array, as are the coordinates for sprites that make up the gamble ladder.

Using the SCREEN\$ command, the fruit sprites are converted to screen strings and put into an array. The whole palette is set to black so you can't see them being converted.

The program then enters the main loop and decides whether or not to allow nudges – you have a one-in-seven chance. Next there's a call to a subroutine that handles the Hold buttons and starting the wheels. In pseudocode it looks like this:

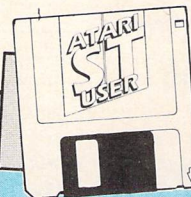
```
IF HOLD available THEN flash HOLD
Start of loop
  Is START selected?
  If HOLD available THEN
    check if any are selected
  ENDIF
End of Loop
```

When the program returns from the

Turn to Page 82 ►

The fruits of your labours

More hints, tips and techniques from David McLachlan in Part 5 of his STOS tutorial



You'll find the complete listing of Fruit Machine on side two of this month's Atari ST User cover disc. You must have STOS in order to run the program.

First boot up your STOS language disc and then insert the cover disc. Load the Basic file as normal – it's stored in the STOS folder. Type RUN to run it.

◀ From Page 81

subroutine 10p is subtracted from your money in the bank and another subroutine sets the machine's wheels spinning.

The fruit sprites on the wheels are read into an array. A pointer is used to indicate the fruit that is to be printed. This done, the pointer is then advanced and the next fruit is printed. That's how the wheels appear to spin.

The pointer for each wheel is only advanced if the *hold* variable for that wheel is set to zero. The wheels are animated using a screen-swapping technique. The screens used are BACK, LOGIC and PHYSIC – their use was explained in detail in our May issue.

Briefly, BACK is the address of the background screen – which is not normally seen. PHYSIC is the address of the physical screen that is being displayed. LOGIC is the logical screen, used in conjunction with the SCREEN SWAP command.

LOGIC is set equal to BACK, which you can't see. When you swap screens PHYSIC and LOGIC are swapped.

Because we made LOGIC equal to BACK the background swaps with the physical screen. The fruits are written to the LOGIC screen so they cannot be seen being printed, then we SCREEN SWAP.

When the wheels have stopped a subroutine is called to find out what amount if any, the winnings are. If you win 20p the variable *gamble* is set to 3, as 20p is the third item up the money ladder.

The sprites either side of the winning amount are animated and the numbers now flash alternately, giving you the chance to increase your money by gambling.

One problem that cropped up during the game's development was how to find out which frame of an animation sequence is currently being displayed. The solution is to DEEK into STOS's sprite work area. First of all you have to find it using the HUNT command like this:

```
code$=CHR$(0)+CHR$(22)+CHR$(0)+
CHR$(122)+CHR$(0)+CHR$(118)
ad=HUNT($1000 TO $ffff,code$)+6
```

This will return the address of the work

area in *ad*. The string *code\$* appears just before the sprite work area – a label if you like. The work area is split up into 14-word sections for each sprite. The frame number currently displayed in an animation sequence is stored in the eleventh word (byte 22). Read using DEEK like:

```
FRAME=deek(ad+(28*spriteno)+22)
```

Another useful DEEK is to find out whether a sprite is onscreen or not. Use HUNT as before, then use the following line of code:

```
sprstat=DEEK(ad+(28*SPRITENO))
```

If the value returned in *sprstat* is zero the sprite isn't on. If it equals one, it is.

The fruit machine program is pretty basic – no pun intended – so why not add extra features yourself?

● Next month I'll show you how to create a short clone of that classic game, Pac-Man.

PASSPORT..STEINBERG..C-LAB..SOUNDBITS

Midi Software for the Atari ST?

NEW LOW PRICES!

Being official main dealers for Steinberg, Dr T, and C-Lab Software means we can offer better prices. Take a look...

Steinberg Pro 24	£249
Steinberg Pro 12	£114
Steinberg Master Score	£264
Dr. T Mrs	£44
Passport MT Junior	£88
C-Lab Creator	£259
C-Lab Notator	£425

Many more items in stock including editors and score printing programs. Call for lists and info!

**BACK IN STOCK – TRACK 24 FOR THE ST,
QUICK BEFORE THEY SELL OUT AGAIN.
ONLY £75**

Best prices in Leeds on DTP, CAD, Graphics and Business Software plus Atari and Amiga Software

**MIDITECH, THE COLOSSEUM,
PORTLAND GATE, LEEDS LS2 3AW.**

0532 446520

MAIL ORDER AND TRAINING
PACKAGES AVAILABLE



TWO WAYS TO ENSURE YOU GET



EVERY MONTH

1. Complete and mail subscription form on Page 109
2. Hand this form to your newsagent.

Please reserve me a copy of Atari ST User magazine every month until further notice.

- ☐ I will collect
☐ I would like it delivered to my home.

Name _____

Address _____

Note to newsagent: Atari ST User should be obtainable from your local wholesaler, or contact Circulation Manager on 0424 430422

TO consolidate the features we have covered so far in this series, on this month's cover disc we have included a fast and addictive arcade game. It's a variation on the Pac Man theme called *Snack Attack*, and it shows just how easy it is to create almost professional quality games using simple STOS commands.

The aim is to guide *Snack Man* to eat all the tablets and occasional random offerings of fruit scattered around a maze while avoiding the creepy ghosts. Your controls are A, Z, < and >. You can get your own back on the nasties by eating the special power pills in each corner of the maze. The ghosts turn blue with fear and you can dispose of them by running them over.

Games usually consist of a main control loop which calls a series of subroutines that perform various actions like drawing the screen, and moving the ghosts or man. *Snack Attack* is no different, and here is its structure in pseudo-code:

```
Start Loop1
  Set up screen
  Check directions
  Start Loop2
    Check for pressed keys and move
    Check directions
    Control ghosts
    Collision detection
    Fruit time?
  End Loop2
End Loop1
```

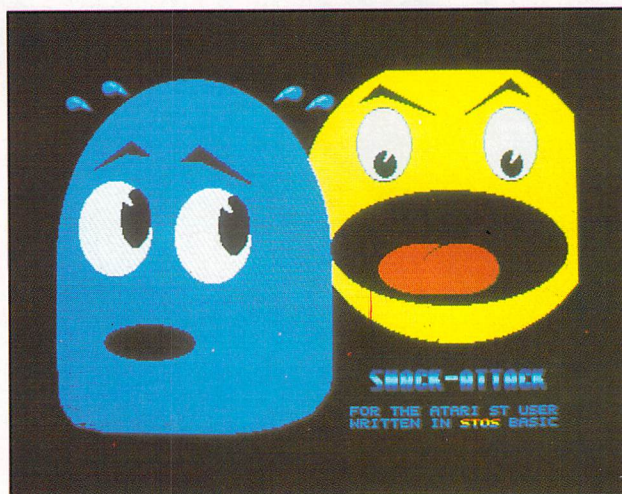
Control ghosts is a subroutine that itself calls other subroutines for each of the three ghosts. These determine the direction in which the ghosts are travelling, which sprites to use, how fast they travel and so on. The maze is read from data statements at the end of the program and stored in memory bank 5.

To move *Snack Man* or a ghost you must first find out what directions are available – there may be a wall in front or behind him. So the subroutine *What directions* is called. By passing the sprite's X,Y coordinate the four direction variables *u*, *d*, *l* and *r* are set to one or zero depending on whether or not a wall is in the way.

The keyboard is checked in the

Keep taking the tablets

Finishing his series on getting the best from STOS, David McLachlan turns to a maze game



subroutine *Check keys* and *Snack Man*'s animation and movement are also set up. The animation strings are defined in the *Set up* routine.

Simple intelligence

A ghost's intelligence is based upon these two simple rules:

- If it's at a dead end turn it around.
- If it comes to a junction it must decide which way to turn to head towards *Snack Man*.

These rules are implemented along with a few exceptions in the subroutine *Find next ghost direction*.

The fruit is controlled by the built-in timer and a marker. When the game starts, the variable *ftime* is made equal to the timer. The simple calculation:

```
timer-FRTIM
```

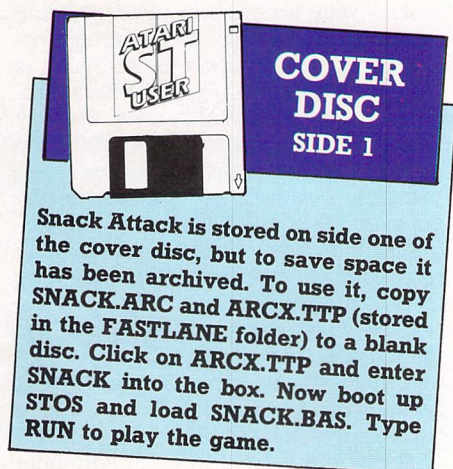
tells you how long the timer has been running since the fruit last appeared or the game started. When the result of this calculation is greater than 1,000 – 20 seconds – a random fruit is selected and

left on the screen for five seconds. After this time if *Snack Man* hasn't eaten it, it is removed from the screen and *ftime* is set equal to the timer again.

The music used is taken from the music files provided with STOS Basic. If you're musically inclined you can create your own tunes using the music editor. I've produced one for when the *Snack Man* dies by using part of the musical scale, copying it to the Cut buffer, pasting it back on to the end of the scale, transposing it by 12 and then repeating the process.

Don't just play the game – although it is very addictive – as there's a lot to be learned by trying to add extra features yourself. *Snack Attack* is fairly basic, so further enhancements like a two player mode or a high score table would improve it.

Also bear in mind that if you have a copy of the STOS Compiler, compiling the program will result in the key responses being keener and the game much faster. And how about adding digitised sound with STOS Maestro, to produce an even more professional result?



Where no man has gone before

HAVING used STOS to write Fun School 2, I've discovered many undocumented features – and a few bugs, along with methods to get around them. It's only fair to share these with other users, so here goes. In this series of articles I'll be showing some of the programming problems you are likely to encounter when writing STOS games, and I'll show how to solve them.

The first problem I'll look at isn't a bug or error, as every version of Basic I know does the same thing. Take a look a Program I:

```
10 REM Program I
20 num=100
30 num$=STR$(num)
40 PRINT num$
50 END
```

You would expect the string `num$` to hold the characters "100", but if you check you'll find that a space is added to the start, so the string is " 100". The simplest way around this is to remove the initial space by changing line 30 to:

```
30 num$=MID$(STR$(num),2)
```

The next problem also concerns the use of **MID\$**. Try entering the following direct commands, that is without line numbers, and see what happens:

```
a$="ABCDE"
a$b$
MID$(a$,3,1)="Z"
PRINT a$
PRINT b$
```

For some reason, which I have yet to work out, altering the middle letter of `a$` also changes the middle letter of `b$`. Have no worries though, as this problem doesn't seem to occur when the commands are executed from within a program.

Going loopy

The next two problems are related to **FOR...NEXT** loops and involve floating point number accuracy – not really a bug, as again many other versions of Basic have this fault. Enter and run Program II:

```
10 REM Program II
20 FOR num=1 TO 2^(4-1)
30 PRINT num
40 NEXT
```

The printed numbers are from one to seven. You may think everything is OK, as the program didn't report any errors, but take a second look at the

Phil Lawson starts a new regular STOS column uncovering some of the language's unusual and unexplored features

end of line 20. Four less one equals three, no problems there, but two to the power of three, ($2 \times 2 \times 2$), is eight. Therefore, line 20 is equivalent to:

```
20 FOR num=1 TO 8
```

So why didn't the loop print out eight numbers instead of just seven? The reason is because the floating point maths routines are used to calculate the loop limit expression and these are not perfectly accurate. Rounding occurs, and some numbers after the decimal point are lost. You can't avoid this, so always bear in mind the limited accuracy of floating point calculations.

The second error with loops concerns nested loops and unexpected errors concerning the use of the **NEXT** statement. Try this example:

```
10 REM Program III
20 FOR a=1 TO 10 : FOR b=1 TO 11
30 PRINT a,b
40 NEXT : NEXT
```

Everything looks all right, but running it gives us a *NEXT without FOR* error. The reason is that STOS expects every **NEXT** command to be followed by a variable name. To get Program III to work change line 40 to:

```
40 NEXT b : NEXT a
```

Some versions of Basic allow the variable names to be omitted, so even if you're proficient at Basic programming read the manual at least once to discover how STOS differs from other Basics.

Spritely commands

As many of STOS's graphics commands concern sprites, it comes as no surprise that one or two peculiarities exist here. Run Program IV – don't type it in, load it from the cover disc to set up the necessary sprites:

```
10 REM Program IV
20 MODE 0:KEY OFF:CURS OFF:HIDE
30 SPRITE 1,0,100,1 : WAIT 50
40 PUT SPRITE 1
50 SPRITE 2,160,0,1 : WAIT 50
60 PUT SPRITE 2
70 SPRITE OFF 1 : WAIT 50
80 SPRITE OFF 2 : WAIT 50
```

First one sprite appears at the left of the screen, followed by a second at the top, then the first one disappears. According to the program (lines 40 and 60) each sprite should have been drawn on the screen before being turned off. Sprite two was in line 60, therefore the problem lies in line 40.

In case you can't guess, the **PUT SPRITE** command doesn't work if the X coordinate is zero. The solution is to alter line 40 to:

```
40 SCREEN COPY PHYSIC,X SPRITE(1),Y
SPRITE(1),X SPRITE(1)+64,Y SPRITE(1)
+64 TO BACK,X SPRITE(1),Y SPRITE(1)
```

Here we're copying part of the physical screen where the sprite is to the background screen – exactly what **PUT SPRITE** does. This, however, will only work if the hot spot is at the top left

Turn to Page 78 ►

◀ From Page 77

corner of the sprite and a multiple of sixteen.(32, 48, 64...).

For this month's final problem we'll take a look at combined screen copy and fade commands. Load and run Program V from the cover disc (don't type it in):

```
10 REM Program V
20 MODE 0:KEY OFF:CURS OFF:HIDE
30 RESERVE AS SCREEN 7 : CLS 7
40 UNPACK 5,7
50 SCREEN COPY 7 TO PHYSIC:WAIT 50
60 FADE 5 : WAIT 35
70 UNPACK 6,PHYSIC
80 FADE 5 TO 7 : WAIT 35
90 GOTO 90
```

What this program is intended to do is display an initial screen, fade it out, and then fade in again with a bit more added to the picture. Unfortunately, the effect of the fade is cancelled out.

To get around this, quite a few changes need to be made:

```
65 LOGIC=BACK : UNPACK 6,BACK
70 FADE 5 : WAIT 35
80 SCREEN COPY BACK,0,160,320,200 T
O PHYSIC,0,160 : LOGIC=PHYSIC : FADE
5 TO 7 : WAIT 35
```

Line 65 uses the LOGIC command to prevent the unpacking cancelling

out the fade. We then unpack the second screen to the background and fade the colours to black. The area required at the bottom of the screen is then copied to the physical screen, which is reset as the logical screen.

The syntax for the SCREEN COPY instruction is:

```
SCREEN COPY screen1,x1,y1,x2,y2 TO sc
reen2,x3,y3
```

In Program V *screen1* and *screen2* are BACK and PHYSIC respectively. This means that the block we want to copy is stored in the background screen, and it will be copied to the physical screen.

The variables *x1* and *y1* mark the top left hand corner of a rectangle, with *x2* and *y2* marking the bottom right. Everything lying within this area will be copied across to the physical screen, with the top left corner placed at position *x3,y3*.

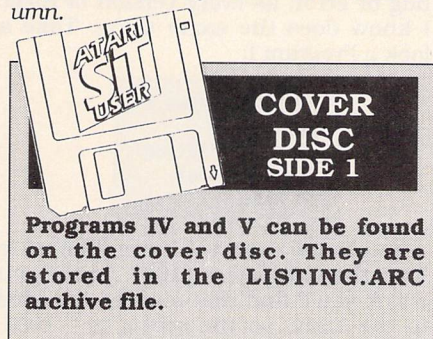
According to the manual, the X coordinates are automatically rounded down to the nearest multiple of 16. This isn't quite true, for as long as the difference between *x1* and *x2* is a multiple of 16 everything will work normally. Note, however, that *x3* will be rounded down.

Now that the screen contains the correct picture all that remains is to

fade back in again. The rest of line 80 sets the logical screen back to PHYSIC and restores the colours with the FADE command.

FADE can turn all the colours to black, but you may not realise how to get them back again. FADE works on the current logical screen, which in our program is now PHYSIC. The number five represents a time delay, so that the colours change at a steady rate, but the TO 7 alters the colours to those in screen 7.

● Next month we'll continue with some more screen effects, and also cock a critical ear to the music and play commands. In the meantime if you think you've found a bug and a way around it, or an interesting effect, write in and I'll pass on the information via this column.



NEW!



CRICIT BAR CODE + PLUS!!!!!!

INVENTORY CONTROL at CASH REGISTER PRICES!

CRICIT BAR CODE + PLUS! is a Bar Code Ready, integrated Cash Register and Inventory Control package. Operations such as sales, adding, editing, deleting and viewing stock are all accomplished by simply scanning bar code found on product packing!

- **Complete cash register** functions
- Flexible **inventory control** for 65,500 products
- **Daily, period and yearly reporting**
- **Price/product labels** with optional bar code
- **Reads bar code directly from packaging**
- **Coupon issue and redemption**(fixed or % of sale)
- **Customized receipts, inventory and reporting**
- **Ready-to-mail purchase orders** with automatic re-order calculation
- **Commission calculation for 15 sales clerks**
- **Mailing lists** in list and label formats
- **Lay-aways, auto discounts, stock searches**
- **Categories, departments, non-stock items**
- **Between-store reporting via modem**
- **User-friendly** error correction and training manual

See how CRICIT can help your business

SEND FOR DETAILS NOW

*An Easy to Use
Solid Business Solution for
Retail Business*

Desk File Option Set1 Reports Print Set2 Set3									
SALES									
PAYMENT BY >>>>>									
			P	PG	PG	PT	PS	PS	PS
			GODACK	ACC REC	AMEXPRS	M/C	VISA	CHEQUE	CASH
LN#	QTY	STK #	STOCK DESCRIPTION			UNIT	DISCOUNT		TOTAL
1	2	""	COMPUTER WEDDING			149.95	149.95		299.90
2	1	""	COMPUTER CONVOCATION			105.95	105.95		105.95
						SUBTOTAL		405.85	
						TAX		32.47	
						TOTAL DUE		438.32	
						GRAND TOTAL		0.00	
CAT DCNT OFF						BALANCE		438.32	
04/22/87 DCNT:0%						TOTAL DUE		438.32	
INVT:2 TRNS:1						GRAND TOTAL		0.00	

*A Complete Bar Code Ready System
....for the Price of a Cash Register!*

Software Express Ltd • 212-213 Broad Street, Birmingham, England B15 1AY • (021)643-9100

WE examined a few minor problems in STOS last month, and found a couple of interesting screen features. Now we'll continue with more screen effects, and finish with some sound and music.

If you missed last month's article, one of our discoveries was that the SCREEN COPY command does not automatically round all the X coordinates to the nearest multiple of 16 as stated in the manual. As long as the difference between the two X coordinates is a multiple of 16, the values can be anything you want. However, don't forget that the destination X coordinate is reduced to the nearest 16.

This fact can be used for some very impressive horizontal scrolling, which can even be faster than the built-in scroll functions. Text can also be scrolled, which some Amiga users say cannot be done on the ST. Proof coming up —load and run Program I:

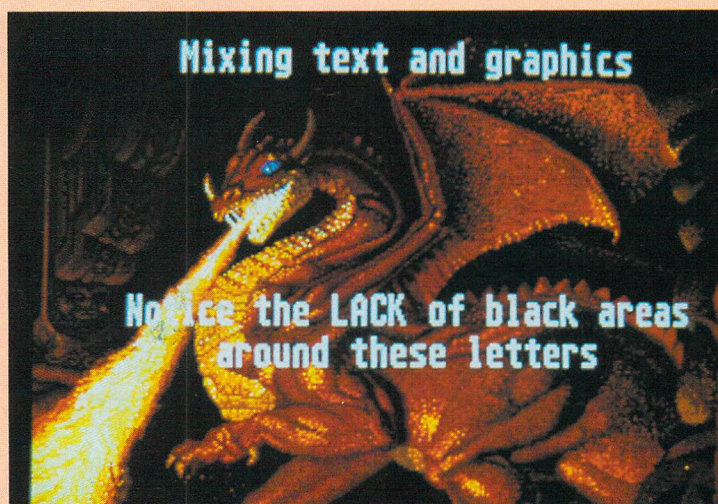
```
10 REM Program I
20 MODE 0:KEY OFF:HIDE:CURS OFF
30 RESERVE AS SCREEN 5:CLS 5
40 RESERVE AS SCREEN 7:UNPACK 6,7
50 GET PALETTE (7)
60 LOGIC=5:WINDOPEN 1,0,0,40,12,0,3
70 RESTORE 160:FOR a=0 TO 4:READ mes
s$
80 LOCATE 0,a:PRINT mess$:NEXT a
90 LOGIC=PHYSIC:SCREEN COPY 7 TO PHY
SIC:WAIT VBL
100 SCREEN COPY PHYSIC TO BACK:WAIT
VBL
110 FOR b=0 TO 4:FOR a=-320 TO 320
120 SCREEN COPY 5,a,b*16,a+320,(b+1
*16 TO PHYSIC,0,175
130 WAIT VBL
140 NEXT a:NEXT b
150 GOTO 110
160 DATA "Nice title scrolling availa
ble in Stos."
170 DATA "Stick your name, or your c
ompanies name"
180 DATA "in a scrolling message for
publicity."
190 DATA "Say HI! to friends and col
legues."
200 DATA "Mention forth-coming produ
cts and games."
```

Explaining all the commands in this program would take a lot of space, so here's a brief summary. Two temporary screens are set up, and the STOS title screen is unpacked into bank seven.

The data messages are read and printed on screen five, each one on a different row. The title screen is then copied to the physical and background screens.

All the hard work is performed by

*When text
is printed
on graphics
you get an
unsightly
black
border*



*This is
easily
solved by
copying
the text
using the
SCREEN\$
function*

Phil Lawson unravels more screen secrets as he continues his STOS series

the FOR ... NEXT loops, which continually copy sections of screen five to the PHYSIC screen. The WAIT VBL command in line 130 makes the scrolling smooth, but also slows it down a bit. Removing this line will result in very fast scrolling, but the text flickers.

Another screen function can be used to place text on a picture screen.

To see this load and run Program II:

```
10 REM Program II
20 MODE 0:KEY OFF:CURS OFF:HIDE
30 WINDOPEN 1,0,0,40,12,0,3
40 CURS OFF:RESERVE AS SCREEN 6
50 UNPACK 5,6
60 SCREEN COPY 6 TO PHYSIC
70 SCREEN COPY PHYSIC TO BACK
80 PEN 15:LOCATE 0,0:CENTRE "Mixing
text and graphics"
90 LOCATE 0,6:CENTRE "Notice the bla
ck areas"
100 LOCATE 0,7:CENTRE "around these
letters"
110 GOTO 110
```

Printing normally puts a rather

Bouncing around

messy black border around the words, which doesn't look very elegant. To get rid of it, make the following changes:

```
75 RESERVE AS SCREEN 7:CLS 7:LOGIC=7
90 LOCATE 0,6:CENTRE "Notice the LAC
K of black areas"
105 LOGIC=PHYSIC:a$=SCREEN$(7,0,0 TO
319,199):SCREEN$(PHYSIC,0,0)=a$
```

All we've done is printed the text to another screen and copied it across using the `SCREEN$` function. What this does is to transfer all the specified area of the screen into a string. The syntax for this command is:

```
a$=SCREEN$(scrn,x1,y1 TO x2,y2)
```

The `x1,y1` and `x2,y2` coordinates are the top left and bottom right positions of the area to store in the string `a$`. The variable `scrn` gets the area from any screen currently in use. To copy the area, the same instruction is used, but in a different way:

```
SCREEN$(scrn,x,y)=a$
```

In this case `scrn` is the screen the area is to be copied to, and the coordinates `x,y` is where the top left corner of the area will appear on the screen.

Looking at line 105, we can see that `a$` is firstly storing the whole of screen seven, which is then copied to the physical screen.

Using this technique rather than the straight forward `PRINT` leaves the background alone, so removing the unsightly black border around the text.

See how a few extra commands can improve the display. It is important to remember these small points when adding the finishing touches to your programs. First impressions really do count.

A musical interlude

STOS, and the ST in particular, are capable of very good music and sound effects, but problems may arise when you try to do both at the same time. What happens is that when a piece of music uses all three sound channels, any sound effects (set with the `PLAY`

command) are obliterated by the music. Let's first take a look at the syntax of the `PLAY` command:

```
PLAY [voice],pitch,duration
```

The *pitch* is the note to be played and *duration* is how long it will last in fiftieths of a second. The *voice* is an optional parameter, and can be used to specify on which sound channel the note will be played. If this is omitted the note will be played on all three channels simultaneously. Section 5, page 103, of the STOS manual explains this and other music commands in greater detail.

Load and run Program III, which is a bouncing ball demonstration:

```
10 REM Program III
20 GOSUB 200
30 DIM c(15),mx$(2),my$(2),an$(2)
40 GOSUB 150
50 SPRITE 1,100,100,1
60 dx=2:dy=1
70 MOVE X 1,mx$(dx):MOVE Y 1,my$(dy)
80 MOVE ON:MUSIC 2
90 WHILE X SPRITE(1)>2 AND X SPRITE(
1)<250 AND Y SPRITE(1)>4 AND Y SPRITE(
1)<130:WEND
100 MOVE OFF:VOLUME 3,15:PLAY 3,30,2
110 IF X SPRITE(1)>2 THEN dx=2 ELSE
IF X SPRITE(1)<250 THEN dx=-1
120 IF Y SPRITE(1)>4 THEN dy=2 ELSE
IF Y SPRITE(1)<130 THEN dy=-1
130 VOLUME 3,0:MOVE X 1,mx$(dx):MOVE
Y 1,my$(dy)
140 MOVE ON:WAIT 10:GOTO 90
150 REM SET DIRECTION MOVEMENT
160 REM *****
170 mx$(1)="(1,-1,0)":mx$(2)="(1,1,0
)"
180 my$(1)="(2,-4,0)":my$(2)="(2,4,0
)"
190 RETURN
200 REM SET UP SCREEN AND COLOURS
210 REM *****
220 KEY OFF:MODE 0:HIDE ON:CURS OFF
230 x=HUNT(START(1) TO START(1)+LENG
TH(1),"palt")+4
240 FOR l=0 TO 15:COLOUR l,DEEK(x):I
NC x:INC x:NEXT
250 RETURN
```

I won't bore you with a complete breakdown of how this program works, but just look at the important bits between lines 60 and 140. Firstly

the directions and movements are set, and the music is started, which uses all three sound channels.

Line 90 waits until the sprite is just about to go off the screen, by testing the X and Y position. Line 100 stops the sprite and plays a note, giving a bounce effect. Lines 110 and 120 alter the direction by checking where the sprite is and reversing the X or Y movement.

The next two lines stop the note, re-assign the direction, set the sprite moving again, and loop back to line 90. The `WAIT` command is there to make sure the sprite has moved away from the edge of the screen before line 90 checks its position again.

If you listen to the sound effects, you'll notice that every now and then the note will be distorted by the music. This is because the music reclaims channel three for its own tunes, giving the note a click sound.

Try changing line 80 to:

```
80 MOVE ON:MUSIC 12
```

This is exactly the same piece of music, but with channel three unused. Now when the note is played it won't be interfered with by the music, which is practically unchanged.

If you wish to see the ball spinning, add the command `SWITCH 5,9` to the end of line 80. This waits for 5/50ths of a second and then rotates all the colours, beginning with number nine. Try changing these two numbers for interesting effects.

You may have noticed that the music is quieter than the bounce note. This is quite useful, as your programs can now have a nice background tune, while the sound effects _ whizzes, bangs and whoopies _ appear to be even more spectacular. To accomplish this make the volume of music about 10 and the sound effects 14 or 15.

● Next month we'll examine some of the accessories, and show a useful technique for reducing the size of sprite banks, without reducing the size or number of sprite patterns.

to a lively tune



Getting on with the

MANY of us have ideas on what makes the perfect arcade game, but just how do we start transforming ideas into actual programs? Starting this month I'm going to be producing a simple arcade game, explaining each of the major steps along the way.

In case you're wondering about the sprite techniques I mentioned last month, this information can be found in a panel on page 108. The program used to demonstrate the effect is called SPRITES.BAS and is on the cover disc in the STOS folder.

Also in this folder is a new set of lo-res characters called 8X8.CRO. Erase the file with the same name from your backup of STOS, but NOT the original disc, and copy this file on to it. As readers have asked for more tips and programming techniques, I've postponed the examining of the accessories for a couple of months.

Down to business

Before we go anywhere near the computer we must plan what type of game to write and the features it is to include. For the purposes of the article I decided on a Bomber game, but as the final stages will be put together around Easter we'll give it a seasonal flavour.

The main program will consist of six columns of goblins and a plane flying overhead. Each goblin is working for

Phil Lawson shows how a programmer's mind works when writing the next chart topper

The game's main features

- Four levels
- Hi-score tables
- Backdrop picture
- Continuous music
- Snappy sound effects
- Cheat mode
- Scrolling text on title screen

the Easter Bunny and will only go away when he gets a chocolate egg dropped from the plane above. When each goblin has been supplied with an egg, the plane lands and you'll progress to the next level. If you crash into a goblin one of your three lives will be lost and you'll have to start the level again.

Right. We've got the idea, but what about the features? After thinking about this for while, I came up with the seven shown in Table I.

Before starting programming we'll

have to design sprite patterns for the plane, goblins and the falling egg. Luckily for you, all the graphics have been done and you'll find them in the program BOMMIT.BAS. Feel free to change these in any way you wish, but do keep a backup set for security reasons.

I mentioned earlier the major steps, which are for this game are:

- Writing the main body of the program.
- Adding the levels and minor routines.
- Music, sound effects and cheat modes.
- Putting on a title screen.

These are fairly brief, and most contain problems that are not immediately obvious. In true programming style, we'll solve each as we come to it, which probably means some routines may need altered later.

This month we'll work on the first point - writing the main body. Loading the program BOMMIT.BAS will give you the program and graphics, but I suggest you remove the lines by entering:

```
DELETE 10-9999
```

You can now follow the article and enter each line as we discuss it. Lines 10000 onwards should be of help to readers who've written in asking how to get the colours from the sprite bank. Don't worry too much about how this routine works, just accept that it does.

game

Begin programming

The easiest way to explain how the program came to its present form is to take you through each of the steps I encountered when I wrote it.

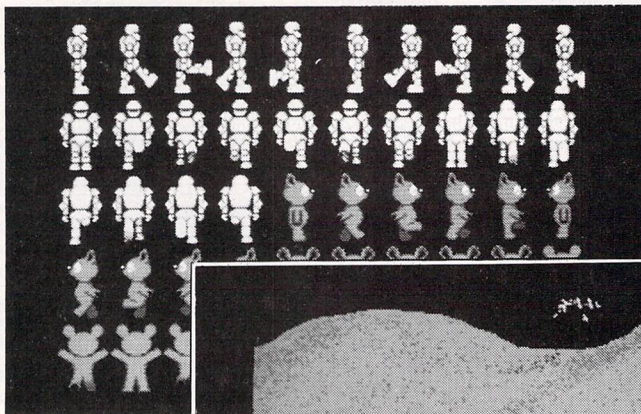
The first part was clearing the screen and getting the colours from the sprite bank. Since this routine will only be called once or twice at the most, I placed it at 10000, well away from the rest of the program.

The first part of the program, which calls all the other subroutines, looks like:

```
10 REM Easter Bomber Game
20 REM by Phil Lawson / Atari ST User
30 GOSUB 10000
90 GOTO 90
```

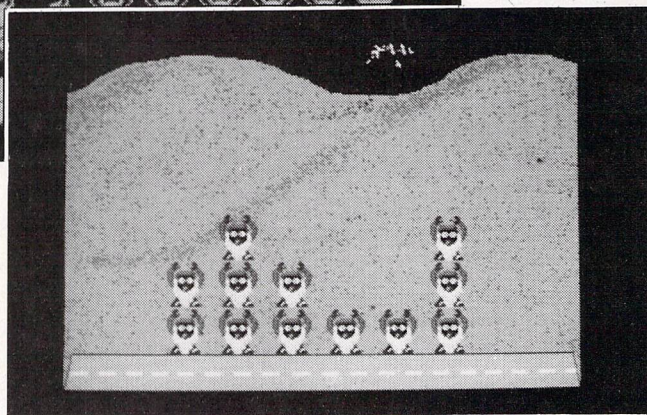
Once the screen is cleared and the colours set up, it's necessary to draw the goblins on the screen. We also need to know their position, which is best held in an array. Since several other variables will also be required, it's best to define these in an initialisation routine, starting at line 500.

```
40 GOSUB 500
500 REM Initialisation
501 REM *****
510 DIM scr(6,6)
520 lev=1
590 RETURN
```



A screenful of animation sequences

Easter Bommit-
The game
so far



The lines to draw the goblins are:

```
50 GOSUB 9000
9000 REM draw the screen
9001 rem *****
9010 FOR a=1 TO 6 : b=RND(lev+1)+1 :
FOR c=1 TO b : scr(a,c)=1
9020 SPRITE 1,32*a+32,180-(26*c),1 :
WAIT VBL : PUT SPRITE 1 : WAIT VBL
9030 NEXT c : NEXT a : RETURN
```

The value of *b* in line 9010 is how many goblins are to be in column *a*. The maximum is 2 plus the level number and is calculated by the function $RND(lev+1)+1$.

Running the game draws the goblins, but not the background screen.

This is achieved with line 9005, but first our initialisation routine will have to unpack the picture from bank 5 and store it ready to be copied on to the main screen.

```
530 RESERVE AS SCREEN 6 : UNPACK 5,6
9005 SCREEN COPY 6 TO PHYSIC : WAIT
VBL : SCREEN COPY PHYSIC TO BACK :
WAIT VBL
```

This finishes the drawing routine, for the time being at least. Later we'll tidy the picture up a bit, but now we can

Turn to Page 28 ►

◀ From Page 25

set to work on the main part of the program, controlling the plane and dropping eggs.

The plane is defined by sprite patterns 2,3,4 and 5, with the hotspot at the bottom right corner. Every time the plane leaves the right side of the screen it has to reappear on the left, but a little bit lower down. To easily test this *y*-coordinate we should use a variable, in this case *ypos*.

As the height of each goblin is 26 pixels, it seems reasonable to decrease the planes' altitude by half of this amount, that is 13 pixels. This, like all the values in the game, can always be altered at a later date if necessary.

To stop the player from dropping eggs all over the place, so making the game too easy, only one egg can be dropped at a time. This means a variable is needed to test if an egg is already falling, before the program releases another one. Line 540 defines the initial values of these two important variables.

```
540 ypos=20 : eggs=0
```

The routine starting at 1000 sets up the animation sequence and controls the movement of the plane.

```
60 GOSUB 1000
1000 REM start playing the game
1001 rem *****
1010 WHILE ypos<190
1020 SPRITE 1,0,YPOS,2 : ANIM 1,"(5,3)
(4,3)(3,3)(2,3)L" : ANIM ON : WAIT VBL
1030 MOVE X 1,"(1,1,350)E350"
1040 MOVE ON 1
1050 WHILE MOVON(1)
1060 WEND
1070 ypos=ypos+13:WEND
1999 RETURN
```

Running the program now flies the plane from one side to the other and makes it reappear at the left 13 pixels further down the screen, which continues until the plane reaches the runway.

Line 1010 is only temporary, as when playing normally the plane will never reach the bottom of the screen. Either all the goblins will be removed or the plane will crash into one of them. We're only using this line to test the rest of the program and we'll change it when everything else works.

The next step is to drop an egg when the spacebar is pressed. After checking there's no egg already on the screen, all we have to do is place one directly under the plane and set it falling. This is achieved with the lines:

```
1055 IF INKEY$=" " THEN GOSUB 2000
2000 rem drop an easter egg
2001 rem *****
2010 IF egg>0 THEN RETURN
2020 SPRITE 2,XSPRITE(1)-16,YSPRITE(1)
,6 : MOVE Y 2,"(1,1,0)E200" : ANIM 2,"
(6,3)(7,3)(8,3)(9,3)L" : ANIM ON 2
2030 WAIT VBL : egg=1
2040 MOVE ON 2 : RETURN
```

We can now drop one egg, which

Spritely action

OWNERS of standard 520STs may well find the maximum sprite bank size of around 30000 bytes a little limiting, especially if their multi-level arcade game requires different characters for each level. However, this problem can be solved quite nicely by using the GET SPRITE function.

Load and run SPRITES.BAS from the cover disc. First a robot walks across the top of the screen, followed by a teddy bear. If you look at line 270 – the section that defines the animation – you'll see that both sprites use the same series of patterns, one to five. How is this possible you may ask?

Bank five, unpacked to screen six, containing more than 40 animation sequences for the robot and teddy bear, and the sprite bank holds five empty patterns. Before we can show the robot walking, we need to copy the patterns from screen 6 to the sprite bank.

This is carried out in lines 70 to 90 with the GET SPRITE command, the syntax of which is:

```
get sprite xpos,ypos,image
```

The coordinates *xpos,ypos* define the top left corner of the area to grab, and *image* is the number of the pattern to be loaded. The size of the image is read from the original dimensions you specified in the sprite bank, and in this

example is 36 pixels wide and 32 high.

The image grabbed is taken from the background screen, hence the need for the commands setting BACK and PHYSIC. The X and Y coordinates have already been initialised in the arrays *robotx* and *roboty*, so all the hard work is done in line 80 with a simple FOR... NEXT loop. The last line restores PHYSIC and BACK, and positions the sprite just off the left-hand side of the screen.

Lines 100 calls the subroutine at 160, which sets the sprite animating and moving. It waits until movement has stopped, just off the right side of the screen. The routine between 110 and 140 is exactly the same as 70 to 100, except the patterns for the teddy bear are loaded into the sprite bank.

Add the following lines to see that the whole screen is full of different patterns for making the robot and teddy walk left, down and up as well as right:

```
45 screen copy 6 to physic
46 goto 46
```

The data in lines 240 and 250 are the x and y coordinates for the images to grab, so you can experiment by changing the movement directions, the x and y data, and adding a few more empty patterns to the sprite bank.

falls to the bottom of the screen and disappears. To find out if it has been dropped directly above a goblin. We have to define a maximum and minimum *x*-coordinate for each column. This allows the egg to be dropped a little to either side of a goblin, rather than right in the middle. Change line 510 and add line 550.

```
510 DIM scr(6,6),ma(6),mi(6)
550 FOR a=1 TO 6 : mi(a)=32*a+34 : ma(a)=32*a+44 : NEXT a
```

Rather than having to continually check whether the egg will hit, we can alter line 2030 to set *egg* to one if it will, or two if not.

```
2030 egg=1 : xb=ABS((X SPRITE(2)-32)/3
2) : tb=X SPRITE(2) : IF tb>ma(xb) or
tb<mi(xb) THEN egg=2
```

The important bit of this line is calculating the value of *xb*, which will be in the range of zero to seven. A value of zero means the egg was dropped at

the left side of the screen, seven means the right side and 1 to 6 means it was dropped near that particular column.

The variable *tb* is the *x*-coordinate of the egg, which is checked against the maximum and minimum values for the column *xb*. If you run the program now everything seems all right, however, dropping the egg at the right side of the screen stops the program with a *Subscript out of range* error.

This is because the value of *xb* is seven. Looking at line 510 shows we've only reserved six elements for the two arrays, so change the last part of this line to:

```
ma(7),mi(7)
```

We'd better set a maximum value for array seven, so the egg will never be dropped with only half of it showing on the screen. Add to the end of line 550:

```
550 ..... : mi(7)=260
```

and alter line 2010 to:

```
2010 IF egg>0 or X SPRITE(1)>270 THEN
RETURN
```

Subroutine functions

500 – Set up arrays and variables
1000 – The game's main routine
2000 – Drop an easter egg
9000 – Draw the screen
10000 – Clear screen and get colour from the sprite bank

We have now finished the routine to drop the egg, and all we have to do is check if the value of *egg* is one and has reached a goblin. If it has, the goblin and egg need to be removed from the screen and we'll tackle this problem next month.

Into the goblin zone

AFTER sorting out some of your programming problems this regular STOS column is now exploring the different effects and techniques used in writing whizz-bang arcade games. STOS is an ideal language for this task and has a whole host of essential commands and functions. The hard part is not coming up with the game idea, it's stringing together those commands to make the sprites and graphics mesh to make the game work.

Last month we wrote the basics of a Bomber game and the program was included on the cover disc. I challenged you to find a way of detecting when a goblin caught an egg. Extra brownie points if you thought about using zones to solve this problem.

Zones are a simple way of detecting when a sprite enters a pre-selected area of the screen. They let us check for collisions with stationary objects without having to know the sprite's coordinates. Up to 128 separate zones can be created, each identified by a number in the range 1 to 128.

To check if the egg has been caught we have to place a zone around each goblin and use the command:

```
t=ZONE(sprite)
```

The variable *sprite* can range from 0

Phil Lawson continues his tutorial on how to write games in STOS

to 15, with the mouse being indicated by zero. If a sprite is in a particular zone, the returned value of *t* is that zone's number. Otherwise, the sprite is not in a zone and *t* will be zero. That's got the theory out of the way, so let's get down to programming.

Every time a goblin catches an egg, it has to vanish from the screen, taking the egg with it. Therefore when they have all gone, the screen is empty and that level has been finished. The simplest way to check if all the goblins have gone is to use a variable which holds the number of goblins on the screen. It should come as no surprise that this variable is called *goblins*.

Change line 9010 and add line 9025:

```
9010 goblins=0 : FOR a=1 TO 6 : b=RND
(lev+1)+1 : FOR c=1 to b : scr(a,c)=1
9025 INC goblins : SET ZONE goblins,3
2*a+32,180-(26*c) to 32*a+64,206-(26*
c)
```

Initially we set *goblins* to zero and every time a goblin is drawn on the screen (line 9020) the variable is increased by one and a zone is set up around it. The syntax of the SET ZONE command is:

```
SET ZONE n,x1,y1 TO x2,y2
```

The zone number to be created is given by *n* and the rectangular screen area covered by the zone is defined by the top left and bottom right coordinates *x1,y1* and *x2,y2*.

We have now achieved two objectives - setting a rectangular zone around each goblin, and calculating the number of goblins.

Last month we set up the variable

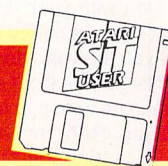
egg, which has three possible values:

```
egg=0 - No egg dropped
egg=1 - Egg dropped on target
egg=2 - Egg dropped but not on target
```

Add the following lines to last month's program to make sure the egg is on target before branching to the routine where the goblin catches it:

```
1056 IF egg=1 THEN GOSUB 3000
3000 REM Check if egg hits goblin
3001 REM *****
```

Before we write the routine we'd bet-



**MONTHLY
DISC
SIDE 1**

ON the monthly disc available to subscribers you'll find the program we developed last month, so all you have to do is enter the lines given here. As the object of this series is to help you become more confident at programming, feel free to alter any of the graphics and screen effect routines, but do keep a backup copy or you won't be able to follow these articles.

ter decide exactly what it has to do.

- 1) Check if egg has been caught
- 2) Remove the egg and goblin from the screen
- 3) Decrease the number of goblins
- 4) Update the scr(x,y) array
- 5) Set the variable 'egg' to zero

Enter the following lines and try to match the various commands with the five points above:

```
3010 z=ZONE(2) : IF z=0 THEN RETURN
3020 yb=(200-YSprite(2))/26 : SPRITE
OFF 2
3030 SCREEN COPY 6,xb*32+32,180-(26*y
b),xb*32+64,206-(26*yb) to PHYSIC,xb*
32+32,180-(26*yb)
3040 SCREEN COPY 6,xb*32+32,180-(26*y
b),xb*32+64,206-(26*yb) to BACK,xb*32
+32,180-(26*yb)
3050 DEC goblins
3060 BELL : WAIT VBL : scr(xb,yb)=0 :
egg=0
3070 RESET ZONE z:RETURN
```

Line 3010 checks if the egg is in a zone and has therefore been caught. If it isn't the program returns to line 1060.

When we wrote the "drop egg" routine last month, the variable *xb* was set as the column number. We now need to find the row number of the goblin surrounded by zone *z*, so we can use both values to remove the goblin from the screen. Line 3020 puts this number into *yb* and turns off the egg sprite.

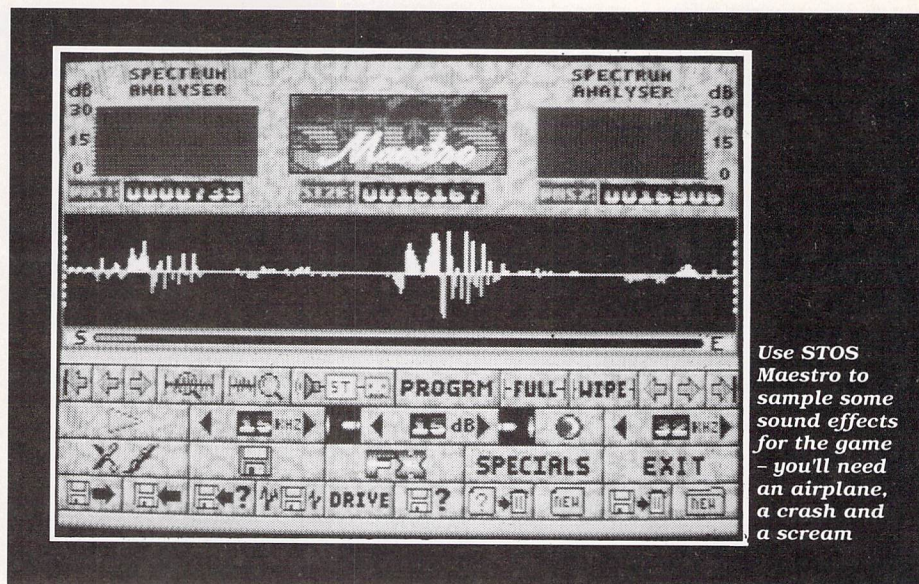
According to point two, the next stage is to remove the goblin. Since the game has a background picture, it would seem reasonable to copy part of this image, stored in bank six, to the physical and background screens as done in lines 3030 and 3040.

Decreasing the number of goblins and resetting the values of *egg* and *scr(x,y)* is accomplished with lines 3050 and 3060, with a little sound effect thrown in for good measure. As the goblin has been removed, there's no need to have its zone still functioning, so line 3070 resets it and returns back to line 1060.

Bug hunting

Check everything works OK by running the program. You should find that several goblins are drawn, the plane moves from left to right and pressing the spacebar will drop an egg. If the egg is on target, when it reaches a goblin both should disappear from the screen, otherwise it will continue until it reaches the bottom.

When the egg misses the goblins and falls to the bottom of the screen the program doesn't allow any more eggs to be dropped from the plane, because the value of *egg* is still two. When we first set the egg sprite falling, we specified the sprite should stop



Use STOS Maestro to sample some sound effects for the game - you'll need an airplane, a crash and a scream

moving when it reached Y coordinate 200. Break into the program and add the following line:

```
1057 IF MOVON(2)=0 AND egg=2 THEN egg =0
: BOOM
```

If *egg* is two, the dropped egg will miss the goblins and eventually stop moving when it goes off the bottom of the screen. The command MOVON tests the state of a given sprite, in this case number two and reports whether it is moving or not. If the returned value is -1 the sprite is still moving, but a value of zero means it has stopped so we can now reset the value of *egg* and produce an appropriate sound effect.

You may have spotted another slight bug when playing the game. If you clear a column of goblins and drop an egg in this column, the value of *egg* will be set to one, meaning that it is on target. However, as there are no goblins in this column, it is definitely not on target and *egg* should be two instead.

If we alter the minimum X coordinate for a column once it has been emptied, any egg dropped in that column will be outside the maximum and minimum values, and so set to a type two egg rather than type one.

To do this alter line 3060 to:

```
3060 BELL : WAIT VBL : scr(xb,yb)=0 :
egg=0 : IF yb=1 THEN mi(xb)=260
```

The final bug only appears when a level is finished and you start the next one. The *scr(x,y)* array and the value of *egg* aren't reset between levels, so may not be able to drop any bombs, and even if you did the zones and array indicate the goblins are in different positions.

Add the following lines to get around these irritating problems:

```
70 GOSUB 540
9002 FOR a=1 TO 6 : FOR b=1 TO 6 : scr
(a,b)=0 : NEXT b : NEXT a
9003 RESET ZONE : SPRITE OFF
```

End games

We have now written all the routines to handle the flying plane, falling eggs and removing the goblins. All that remains for this month is to add the different levels and lives. To do this make the following changes:

```
45 WHILE livesgtr0
90 WEND
99 STOP
520 lev=1 : lives=3
1005 done=0
1010 WHILE done=0
1058 IF ZONE(1)ltngtr0 THEN done=2 :
ANIM OFF : MOVE OFF : DEC lives
3050 DEC goblins : IF goblins=0 THEN
done=1
```

In the above lines you'll see the variable *done* which can be one of three values:

```
done=0 Still playing the game
done=1 Every goblin has been removed
done=2 The plane crashed into a goblin
```

In the original program line 1010 used the plane's Y position to control the WHILE...WEND loop. We have now changed this so the loop will continue until either the plane crashes or all the goblins have gone.

Line 1058 uses the ZONE command to see if the plane has hit a goblin. If it has, the variable *done* is set to 2 and the number of lives is decreased. The program will now return to line 90 and if the player has any remaining lives the game will start again at the current level, otherwise it's "Game Over" time.

The actual crashing and Game Over messages will be tackled next month, but when a level has been completed we'll make the plane land on the runway. This is simply achieved by putting the plane at the bottom-left side of the screen, make it fly right for a bit and then gently lower it onto the

tarmac. Since this routine will only be used when a level has been completed, it's a good idea to include the section of code which increases the level number, as done in line 1570.

```
1080 IF done=1 THEN GOSUB 1500 ELSE DE
C lives
1090 RETURN
1500 REM Land the plane
1501 REM *****
1510 ypos=170 : SPRITE 1,0,ypos : MOV
E X 1,"(1,1,100)" : MOVE ON 1
1520 WHILE MOVON(1) : WEND
1530 MOVE X 1,"(1,1,50)" : MOVE Y 1,"
(2,1,25)" : MOVE ON 1
1540 WHILE MOVON(1) : WEND
1550 MOVE X 1,"(1,1,50)" : MOVE ON 1
1560 WHILE MOVON(1) : WEND : ANIM OFF
1570 IF lev<4 THEN INC lev
1580 RETURN
```

To finish off this month we'll include a few commands to make the screen fade in and out.

```
1580 FADE 5 : WAIT 35 : CLW : SPRITE
OFF
1590 RETURN
9001 FADE 5 : WAIT 35 : CLW : SPRITE
OFF
9005 SCREEN COPY 6,0,0,320,200 TO PHY
SIC,0,0 : WAIT VBL : SCREEN COPY PHY
SIC TO BACK : WAIT VBL
9030 NEXT c : NEXT a : WAIT 50 : FADE
5 TO 6 : WAIT 35 : RETURN
```

Hints and tips

This month's selection of hints and tips comes from Ian Hayes.

After using the Play command to produce a sound, the note can often still be heard even when the program has finished. To get around this slight irritation, issue the command PLAY 0,0 to stop the sound.

When building screens out of sprites with the PUT SPRITE command, occasionally it wouldn't work even when preceded it with plenty of WAIT VBL statements. If the X coordinate of the sprite is zero you'll need to copy the sprites area from the physical screen to the background. In other cases try using the UPDATE command, especially if you intend to compile the program at a later date.

After changing the mouse and defining a menu, the mouse is

reset to the normal arrow pointer. I have found no answer to this problem except to define the menus before changing the mouse.

The last tip also concerns menus. Turning them on and off is easily achieved with the MENU FREEZE and MENU ON commands, but freezing a menu often leaves the menu bars at the top. To get rid of this try the line:

```
INK 0 : BAR 0,0 TO 319,7
```

If you're working with a detailed screen you could always use the SCREEN COPY function.

● If you'd like to see your name in print drop me a line. Each month the sender of the best tips will receive a top quality game.

● Next month we'll write the crashing and music routines and also put the score, level and lives at the top of the screen.

CLUB 68000

CLUB 68000, Wickham House, 2 Upper Teddington Road, Hampton Wick, Kingston, Surrey KT1 4DP. Tel: 01-977 9596. Fax: 01-977 5354

Club 68000 offers members, software, hardware and accessories at huge savings off recommended retail prices. When you join you will receive a free games compendium and a catalogue five times a year. Your only commitment is to pay £15 for one years membership. Please post to the address above.

ATARI ST SPECIALIST

TOP 40 GAMES

	R.R.P.	Members
Batman The Movie	£19.99	£12.99
Beverly Hills Cop	£24.99	£15.99
Bomber	£29.99	£19.99
Blade Warrior	£24.99	£15.99
Cabal	£19.99	£12.99
Chaos Strikes Back	£19.99	£12.99
Chase HQ	£19.99	£12.99
Chess Player 2150	£24.99	£15.99
Continental Circus	£19.99	£12.99
Double Dragon II	£24.99	£15.99
Falcon F16	£19.99	£12.99
Falcon Mission Disk	£24.99	£15.99
F16 Combat Pilot	£24.99	£15.99
Galaxy Force	£24.99	£15.99
Ghostbusters II	£24.99	£15.99
Ghost 'n' Ghouls	£24.99	£15.99
Hard Drivin	£24.99	£15.99
Honda RVF	£24.99	£15.99
Indiana Jones Last Crusade	£19.99	£12.99
Infestation	£24.99	£15.99
Interphase	£24.99	£15.99
Knight Force	£24.99	£15.99
Light Force (4 Games)	£24.99	£15.99
Moonwalker	£24.99	£15.99
North And South	£24.99	£15.99
Onslaught	£24.99	£15.99
Operation Thunderbolt	£19.99	£12.99
Paul Gascoigne Soccer	£24.99	£15.99
Populous	£24.99	£15.99
Powerdrift	£24.99	£15.99
Pro Tennis Tour	£19.99	£12.99
Shadow of the Beast	£34.99	£24.99
Space Ace	£44.99	£31.99
Stryder	£19.99	£12.99
Stryx	£19.99	£12.99
Stunt Car	£24.99	£15.99
Turbo Outrun	£19.99	£12.99
Untouchables	£24.99	£15.99
Wildstreet	£24.99	£15.99
Xenon II	£24.99	£15.99

TOP 40 PROFESSIONAL

	R.R.P.	Members
1st Word Plus	£79.99	£55.99
Cubase	£499.00	£399.00
Degas Elite	£24.99	£15.99
DG Base	£34.99	£49.99
DG Calc	£39.99	£27.99
Fontz	£29.99	£19.99
Fun School 2	£19.99	£12.99
Home Accounts	£24.99	£17.99
Hisoft Basic	£79.99	£59.99
Hisoft C	£49.99	£34.99
Hisoft Devpac 2.0	£59.99	£39.99
K Comm II	£49.99	£34.99
K Data	£49.99	£34.99
K Graph III	£49.99	£34.99
K Index	£29.99	£20.99
K Minstrel 2.0	£29.99	£20.99
K Resource II	£39.99	£24.99
K Rikki	£19.99	£12.99
K Rhymes	£19.99	£12.99
K Seka	£49.99	£34.99
K Spread II	£49.99	£34.99
K Spread III	£89.99	£62.99
K Switch II	£29.99	£20.99
Mastersound Digitizer	£39.99	£29.99
Mini Office Comm.	£24.99	£15.99
Mini Office Graphics	£24.99	£15.99
Mini Office Spreadsheet	£24.99	£15.99
Pro 12	£129.99	£55.00
Pro 24	£299.99	£199.99
Skyplot	£69.99	£49.99
STOS	£29.99	£19.99
STOS Compiler	£19.99	£12.99
STOS Maestro	£24.99	£15.99
STOS Maestro+	£69.99	£49.99
STOS Sprites 600	£14.99	£9.99
Superbase Personal II	£99.99	£69.99
Superbase Professional	£249.99	£174.99
Turbo ST	£39.99	£24.99
Virus Protector	£14.99	£9.99
Wordup	£59.99	£39.99

HARD DISKS & DISK DRIVES

● 3.5" 720K DRIVE, Incl. Power supply through port	£79.99
● 3.5" With Digital Track Display	£94.99
● 3.5" 720K DRIVE, Internal	£64.99
● 5.25" 40/80 Track Drive, IBM compatible	£99.99
● 5.25" With Digital Track Display	£114.99
Members	RRP
● Megafile 30 MByte	£369.99 £499.00
● Megafile 60 MByte	£599.00 £699.00

NEW PC SPEED MS DOS Hardware Emulator

● Runs MS DOS software fast and bug free	
● Norton factor 4.0 (4 times faster than 4.77 MHzxt)	Members RRP
● IBM Monochrome and CGA Hercules emulator	£249.00 £379.00

ACCESSORIES

ELECTRONIC MONITOR SWITCHBOX	£29.95
● Switch between monochrome & colour monitors	
● Also switching without reset!	
CLOCK CARTRIDGE	£29.95
● Realtime version	

First impressions

FOR the next few months I will be examining a variety of different subroutines commonly used in arcade games which can be added to your own listings. I'll start by examining the various methods of displaying a title screen. Future articles will cover high score tables, cheat modes, sprite routines and many other features that no game should be without.

All the programs shown here can be found in the LISTINGS folder on this month's *Atari ST User* disc. Alternatively draw your own title screen with an art package, compact it and store it in memory bank five. Bank six should also contain a compacted screen of

Phil Lawson takes a look at different methods of displaying title screens in your arcade and adventure games and provides several ready-to-use subroutines

text, and non-subscribers should design their own, with a routine similar to Program I (below).

The saved text picture should be compressed using COMPACT.ACB, but first you could add some interesting features to the letters with an art package such as Degas Elite. When

designing a title screen it's best first to write a routine which displays all the game's sprites and saves the whole screen as a picture file. This can then be edited and changed using an art package to give a professional, high quality appearance.

The different ways of displaying title screens we'll be covering are just a few of the many effects possible. I'll leave it up to you to discover others and include your own animation sequences, although it's worth bearing in mind that subtle effects can be achieved by simple palette switching with the SHIFT command.

Enough of the chat, let's get down to business. Of all the STOS programs that flood into the office each month the majority tend to rely on the familiar FADE and APPEAR commands to switch between two screen displays. While both give attractive results, they can bore after a while. The next two

```
10 REM Program I
20 REM Create text screen
30 MODE 0 : KEY OFF : HIDE
40 WINDOPEN 1,0,0,40,12,0,3 : CURS 0
FF: PEN 14
50 FOR a=0 TO 11 : LOCATE 0,a
60 READ mess$ : CENTRE mess$
70 NEXT a : WAIT 100
80 SAVE "MESSAGE.PI1",BACK
90 GOTO 90
1000 DATA "Build your title screens f
rom sprites","used within the game."
1010 DATA "Try different ways of disp
laying the","title screen for impressi
```

ve effects."

```
1020 DATA "Use scrolling text and fla
shing colours","for displaying instruc
tions."
```

```
1030 DATA "Don't forget to buy Atari
ST User for","STOS programming hints a
nd tips."
```

```
1040 DATA "Better still, take out a s
ubscription.,""Send in your hints and
tips."
```

```
1050 DATA "The best each month wins t
wo top games.,""Look out for Games Gal
ore and TOME."
```


programs show simple ways of adding spice these effects:

```
20000 REM Title screen #1
20010 MODE 0 : KEY OFF
20020 CURS OFF : HIDE
20030 RESERVE AS SCREEN 10 : RESERVE
AS SCREEN 11
20040 UNPACK 5,10 : UNPACK 6,11
20050 SCREEN COPY 11 TO PHYSIC : WAIT
50
20060 FADE 5,$777,$777,$777,$777,$777
,$777,$777,$777,$777,$777,$777,$777,$777,$777
20070 CLW : s$=SCREEN$(10,0,0 TO 320,
200) : SCREEN$(PHYSIC,0,0)=s$
20080 WAIT 50
20090 FADE 5 TO 10 : WAIT 35
20100 GOTO 20100
```

```
20000 REM Title screen #2
20010 MODE 0 : KEY OFF
20020 CURS OFF : HIDE
20030 RESERVE AS SCREEN 10 : RESERVE
AS SCREEN 11
20040 UNPACK 5,10 : UNPACK 6,11
20050 SCREEN COPY 11 TO PHYSIC
20060 APPEAR 10,77
20070 APPEAR 10,20
20080 GOTO 20080
```

The first program displays the message screen, fades all the colours to white, clears the screen, draws the title and fades from white to the correct colours. Most STOS programmers seem to fade the colours to black or white only, but there's no reason why others such as blue, green, yellow and red cannot be used.

The second program snippet implements a little used feature of the



You'll need to draw two title screens like these before running the example listings

Build your title screens from sprites used within the game.
Try different ways of displaying the title screen for impressive effects.
Use scrolling text and flashing colours for displaying instructions.
Don't forget to buy Atari ST User for STOS programming hints and tips.
Better still, take out a subscription.
Send in *your* hints and tips.
The best each month wins two top games.
Look out for Games Galore and TOME.

➤ APPEAR command, the syntax of which is:

APPEAR screen,fade type

If the *fade type* is greater than 72, the final picture is left as a mixture of the original screen display and the one you're attempting to fade in. This allows us to use two different types of fade, one immediately after the other, giving a very pleasing effect. Try altering the fade values in lines 20060 and 20070 and note those which give an interesting result.

Notice that in the above two programs lines 20010 to 20040 are identical. This is true for every program we'll be using this month, so to save space I'll omit them from the listings, but don't forget to type them in yourself.

Scrolled screens

One of the most versatile commands in STOS is SCREEN COPY, which, as its name suggests, copies a defined area of one screen to a given place on another. The syntax of this command is:

SCREEN COPY s1,x1,y1,x2,y2 TO s2,x3,y3

s1 = Source screen
s2 = Destination screen
x1,y1 = Top left corner of area
x2,y2 = Bottom right corner of area
x3,y3 = Top left corner to place area

This opens up the possibility of displaying a title picture one section at a time and scrolling the image over the current display. Run the following program:

```
20000 REM Title screen #3
20050 FOR a=1 TO 200
20060 SCREEN COPY 10,0,0,320,a TO PHYSIC,0,200-a
20070 NEXT a
20080 GOTO 20080
```

The scrolling effect may look rather complex, but it's all controlled with lines 20050 to 20070. Each time round the FOR/NEXT loop, an increasingly larger section of the picture stored in bank 10 is copied to the physical screen. This gives the appearance that the image is being scrolled up from the bottom of the screen. To see it slowly covering an existing picture, add this line:

```
20045 SCREEN COPY 11 TO PHYSIC:WAIT 100
```

Instead of overwriting the original picture we could scroll it off the top of the screen by adding line 20055 and changing line 20060:

```
20000 REM Title screen #3b
20055 SCREEN COPY PHYSIC,0,1,320,199 TO PHYSIC,0,0
20060 SCREEN COPY 10,0,a-1,320,a TO PHYSIC,0,198
```

It's also possible to scroll the picture

Hints and tips

This month's prize of a top quality arcade game goes to Mathew Barry for his tips on expanding the size of the sprite banks.

Since the maximum sprite bank for a standard 520ST is around 30k, some users may feel a little restricted in what they can achieve. One way around this problem, especially for games where different sprites are used on different levels, is to create two sprite files and store one of them in bank two.

To change between sprites all you need to do is use the command:

```
BCOPY 2 TO 1
```

This will copy the contents of bank two into bank one. Obviously it also destroys the first set of sprites, so if they'll be needed later you'll have to copy them first to a spare bank. For instance:

```
BCOPY 1 TO 15
BCOPY 2 TO 1
```

Another advantage of this technique is that separate levels can also have a different range of colours.

down the screen rather than upwards:

```
20000 REM Title screen #4
20050 SCREEN COPY 11 TO PHYSIC
20060 WAIT 100 : FOR a=1 TO 200
20070 SCREEN COPY 10,0,200-a,320,200 TO PHYSIC,0,0
20080 NEXT a
20090 GOTO 20090
```

So far we've scrolled the whole screen either up or down. However, it's possible to scroll the top half down and the bottom half up at the same time:

```
20000 REM Title screen #5
20050 SCREEN COPY 11 TO PHYSIC:WAIT 100
20060 FOR a=1 TO 100
20070 SCREEN COPY 10,0,100-a,320,100 TO PHYSIC,0,0
20080 SCREEN COPY 10,0,100,320,100+a TO PHYSIC,0,200-a
20090 NEXT a
20100 GOTO 20100
```

Scrolling text

You may have noticed that there is a blank area near the bottom of the screen and a screenful of text has been unused. These last two routines will scroll the text within this blank area, first horizontally and then vertically.

To save time I'll simply copy the title picture on to the screen instead of

using some of the fancier techniques we have covered:

```
20000 REM Title screen #6
20050 SCREEN COPY 10 TO PHYSIC:SCREEN COPY PHYSIC TO BACK
20060 FLASH 14,"(777,5)(666,5)(555,5)(444,5)(333,5)(222,5)(111,5)(222,5)(33,5)(444,5)(555,5)(666,5)(777,5)"
20070 FOR a=0 TO 11
20080 FOR b=-320 TO 320
20090 SCREEN COPY 11,b,a*16,b+320,(a+1)*16 TO PHYSIC,0,164
20100 WAIT VBL
20110 NEXT b : NEXT a
20120 GOTO 20120
```

The scrolling text effect is produced with lines 20070 to 20110. Each time around the loop a block of 320 pixels wide is copied from bank 11 to the physical screen. The value of *a* indicates which line of text to display and *b* indicates where the 320 pixel block starts.

Imagine we have two other blank screens by the left and right of bank 11. The SCREEN COPY command will use them to give the appearance of text moving on to the screen from the right side and disappearing off the left.

You may be wondering how this works when the value of *b* is not a multiple of 16. The manual states it will be rounded down to the next lowest multiple, but this is not the case. Providing the width of the area to be copied is multiple of 16 the start and end values, (*x1* and *x2*), can be anything at all.

Sadly this typed text scrolling is not only slow, but can be tedious, especially when waiting for the next line to appear. An interesting way around this is to scroll the text vertically within the blank area of the screen, as shown in our final routine for this month:

```
20000 REM Title screen #7
20050 SCREEN COPY 10 TO PHYSIC:SCREEN COPY PHYSIC TO BACK
20060 FLASH 14,"(777,5)(666,5)(555,5)(444,5)(333,5)(222,5)(111,5)(222,5)(33,5)(444,5)(555,5)(666,5)(777,5)"
20070 FOR a=0 TO 192
20080 SCREEN COPY 11,0,a-16,320,a TO PHYSIC,0,164
20090 IF (a MOD 16)=0 THEN WAIT 100
20100 WAIT VBL
20110 NEXT a
20120 GOTO 20120
```

This works in a very similar way to the last program, except it is the vertical values, (*y1* and *y2*), which are altered. Since the maximum height of each character is 16 pixels, line 20090 waits until the result of *a* divided by 16 is zero and then waits for two seconds.

Instead of scrolling up, the text could be made to scroll down, but I'll leave you to work out how. Also, as a small programming exercise try to combine the last two programs to produce text which scrolls right to left, but also moves slightly up and down.

● *Next month I'll show some of the many ways of entering your name into high score tables, along with a few examples on how to display them.*



LLEFT you last month with the problem of scrolling text both horizontally and vertically, and if you have managed to do it, well done. In programming there is no right or wrong way of doing things. As long as the code works and is easy to follow, almost any method you want can be used to give the desired results.

As I promised this month's article is devoted to high-score routines, displaying the scores and entering players' names.

First, you'll have to design a screen and squash it with the COMPACT.ACB utility.

To help you, Program I will produce the screen I have used throughout this article. It should be compacted and stored in bank five.

You will also have to create a sprite bank of letters and a full stop. This can easily be done with the SPRITE.ACB accessory by grabbing the letters from the picture created with Program I, where sprites 1 to 26 should be the letters A to Z, and sprite 27 being the full stop:

```
5 REM Program I
10 MODE 0 : KEY OFF : HIDE
20 WINDOPEN 1,0,0,40,12,0,3
30 CURS OFF : RESTORE 100
40 FOR a=1 TO 4 : READ mess$
50 LOCATE 0,(a*2) : CENTRE mess$
60 NEXT a
70 WAIT 100
80 SAVE "HIPIC.PI1",BACK
90 GOTO 90
100 DATA "A B C D E F G H I"
110 DATA "J K L M N O P Q R"
120 DATA "S T U V W X Y Z ."
130 DATA "Enter Delete Space"
```

No matter how you decide to display

The name game

Phil Lawson continues
to examine useful
subroutines and turns
his attention to the
high-score table

the table and enter the players' names, you first have to produce a dummy table which contains a few high scores for the players to try and beat which can give the player an incentive.

Lines 21040 and 21050 of Program II set up this initial table, which is displayed with lines 21060 to 21100. I've used a simple print loop, but you could easily substitute this for one of

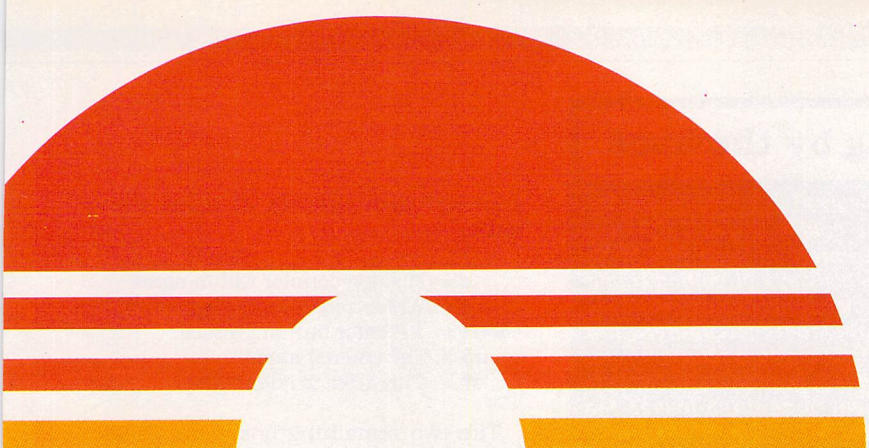
the routines we discussed last month:

```
21000 REM Program II
21010 MODE 0 : KEY OFF
21020 WINDOPEN 1,0,0,40,12,0,3
21030 CURS OFF
21040 DIM hs(10),hs$(10)
21050 FOR a=1 TO 10:hs$(a)="Atari ST
User" : hs(a)=(11-a)*1000 : NEXT a
21055 sc=6543 : GOSUB 21200
21060 LOCATE 0,0:CENTRE "High Score
Table"
21070 FOR a=1 TO 10:LOCATE 18-LEN(hs
$(a)),a+1
21080 PRINT hs$(a);" .....";hs(a);
21090 NEXT a
21100 GOTO 21100
21200 IF sc<hs(10) THEN RETURN
21210 GOSUB 22000 : REM get name
21220 hs$(10)=n$ : hs(10)=sc
21230 FOR a=9 TO 1 STEP -1
21240 IF hs(a+1)>hs(a) THEN SWAP hs$
(a),hs$(a+1) : SWAP hs(a),hs(a+1)
21250 NEXT a : RETURN
22000 n$="Phil" : RETURN
```

The rest of Program II – lines 21200 to 22000 – is concerned with reading the player's name and placing it into the correct high-score position. This section of code is called from line 21055, which also sets a dummy score so you can see the routines working.

Running the above program will show that Phil's score of 6,543 is placed into position in the table. Looking at lines 21200 to 21250 will show how the program decides exactly where the new score should appear. The simplest way of doing this is to use a technique called bubble sorting.

First, the player's score is checked with the tenth or lowest score. If the player hasn't beaten this value, he hasn't made it to the table and the program will return to line 21060. Try



changing the value of `sc` to 900 in line 21055 to see what I mean.

Line 21210 jumps to the routine starting at 22000 to get the player's name. For the time being I've placed a dummy line at this position, but we'll be exploring different ways of doing this in a moment.

The important lines are 21220 to 21250, which first places the new score at position 10. We now check if this score is greater than the next one at position 9.

If it is, the program makes use of the `SWAP` command, which just exchanges the contents of two variables or strings. For example, enter the following commands:

```
A=10 : B=20
A$="YES" : B$="NO"
SWAP A,B
SWAP A$,B$
PRINT A,B
PRINT A$,B$
```

You'll see that the value of `a` is 20, `b` is 10, `a$` is "NO" and `b$` is "YES".

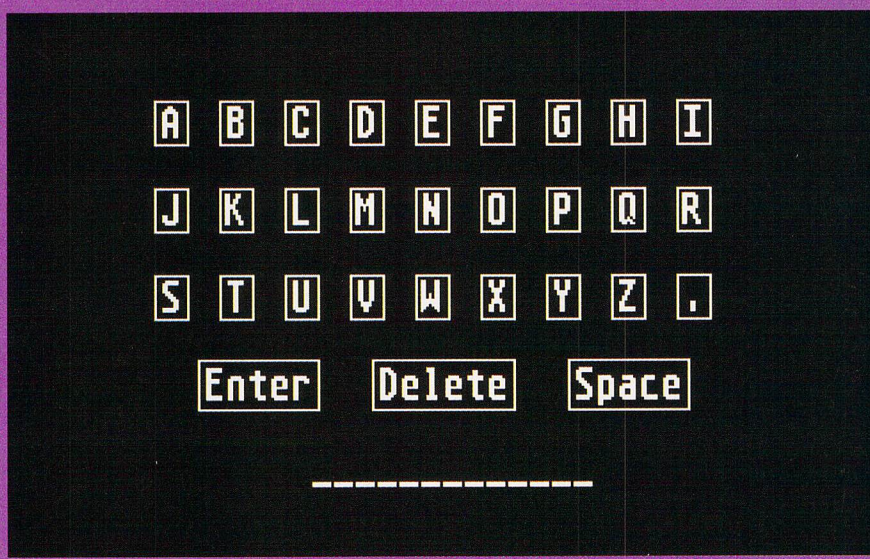
When the player's score is less than the next one in the table, the new name and score will be in the correct place. What happens when two scores are the same? Try changing the value of `sc` to 6,000 and run the program again. You'll notice that Phil's score is placed immediately below the existing one. This is how it should be, as you can only pass a score if you beat it.

Notice that in line 20170 the `LOCATE` command uses the length of each name. This is because the player may only enter a short name, or just his initials, so leaving the high-score table looking rather messy. Try



A B C D E F G H I
J K L M N O P Q R
S T U V W X Y Z .
Enter Delete Space

The output from Program I is used to create a sprite bank of letters and a full stop



A B C D E F G H I
J K L M N O P Q R
S T U V W X Y Z .
Enter Delete Space

Click on the letters, and use delete and space to edit the input

replacing line 21070 with the following line to see this untidy effect:

```
21070 FOR a=1 TO 10 : LOCATE 5,a+1
```

What's in a name?

The first method of inputting a player's name that we'll examine is probably the most common, where the letters of the alphabet are displayed on the screen, and the player chooses which ones to use with the mouse. Add the following lines to Program II:

```
22000 REM Hiscore 3
22010 REM =====
22020 RESERVE AS SCREEN 10 : UNPACK 5,10
22030 SCREEN COPY 10 TO PHYSIC : SCREEN COPY 10 TO BACK
22040 x=1
22050 FOR a=1 TO 3 : FOR b=1 TO 9
22060 SET ZONE x,b*24+30,a*32 TO b*24+42,a*32+16
22070 BOX b*24+30,a*32 TO b*24+42,a*32+16
22080 INC x : NEXT b : NEXT a
22090 BOX 70,127 TO 114,145
22100 SET ZONE 28,70,127 TO 114,145
22110 BOX 134,127 TO 186,145
22120 SET ZONE 29,134,127 TO 186,145
22130 BOX 206,127 TO 250,145
22140 SET ZONE 30,206,127 TO 250,145
22150 SHOW : LOCATE 14,10 : n$=""
      " : PRINT n$
22160 done=0 : c=1 : WHILE done=0
22170 z=0 : IF MOUSE KEY=0 THEN 22170
22180 z=ZONE(0) : IF z=0 THEN 22170
22190 c$="" : IF z<27 THEN c$=CHR$(z+64)
22200 IF z=27 THEN c$=""
22210 IF z=28 THEN done=1
22220 IF z=30 THEN c$=""
22230 IF c$<>"" AND c<>14 THEN GOSUB 22300
22300 : INC c
22240 IF z=29 AND c>1 THEN DEC c : c$="" : GOSUB 22300
22250 WEND
22260 FOR a=1 TO 13 : IF MID$(n$,a,1)="" THEN MID$(n$,a,1)=" "
22270 NEXT a
22280 IF RIGHTS$(n$,1)="" THEN n$=LEFT$(n$,LEN(n$)-1) : GOTO 22280
22290 CLW : RETURN
22300 MID$(n$,c,1)=c$
22310 LOCATE 14,10 : PRINT n$
22320 WHILE MOUSE KEY<>0 : WEND
22330 RETURN
```

Lines 22000 to 22150 are concerned with displaying the screen, defining a zone and drawing a box around each character and option.

Running the program is probably the best way to see how it works. Whenever you click on a letter or full stop, that character is placed in the name and displayed on the screen. Spaces can also be entered by clicking on the SPACE box. Since we have rather a lot to get through this month, I'll not bother explaining exactly how the routine works, but as it is straightforward you should be able to follow it

Going by the book

ANYONE involved in programming for more than a couple of years will have collected a library of useful routines, devised simple ways of getting round difficult problems and discovered many tricks about their computer and favourite programming language.

The rate which professionals can produce efficient code often leaves the novice astounded. Novices often wish they could bypass the tedious learning process and write quality software at the same speed.

Sadly there's no easy way to learn these programming techniques and using other people's routines can lead to great difficulties if you're not entirely sure what's going on. Many books have been published which explain how to write simple routines and now one has been written for STOS programmers.

Stephen Hill, author of the original STOS manual, has collected many interesting routines and combines them with easy to follow text which explains many of the pitfalls of game creation. How to produce fast action sprites, animation, scrolling, 3D graphics and sound are only a few of the topics covered.

As STOS is a true programming language rather than just a construction kit, many different types of games are possible, including shoot-'em-ups, simulations, role playing, rebound games and adventures. This book has chapters devoted to each and explains how best to write them with the help of fully working programs.

I'm glad to say that the book makes use of the free games supplied with each copy of STOS, explaining the various tricks and techniques stored within Orbiter and Zoltar. Producing alien attack waves, deciding the angle at which a ball should bounce, scrolling the screen, getting the best from the map definer and using screen-flipping routines for high-speed graphics are just some of the subjects covered in great detail.

If a particular section of Basic code is too slow, it may be necessary to re-write it in machine code. Programming in 68000 isn't all that difficult, especially as STOS has included many functions to control sprites, windows, sound and every other command available in Basic.

The final chapter contains all the information you'll need to include fancy machine code effects within your Basic program, along with many examples to get you started.

With more than 280 pages packed with vital information and routines, The Game Maker's Manual is a book no STOS user should be without. Stephen Hill deserves full marks for this work.

Product: The Game maker's Manual
Price: £11.95
Supplier: Sigma Press, 1 South Oak lane, Wilmslow, Cheshire SK9 6AR
Tel: 0625 531035
ISBN: 1-85058-158-4

yourself with the aid of these variables:

c = Position counter within name
z = Which zone was clicked on
done = 1 if enter button pressed
n\$ = The entered name
c\$ = Character to add to n\$

The two remaining boxes are Delete and Enter. The first removes the last chosen character, while Enter returns to the line which called the routine, with the name stored in the string n\$. Since the name started off being 13 underline characters (-), lines 22260 to 22280 change these and remove unnecessary spaces at the end of the name.

Instead of the chosen letter automatically appearing at the next position in the name, this alternative version of Program III allows us to pick up a character and place it wherever we wish. Before entering these lines type:

```
DELETE 22160-22330
```

You can now add the new lines which give a more pleasing effect.

```
22000 REM Hiscore 3b
22160 done=0 : WHILE done=0
22170 z=0 : WHILE z=0 : WHILE MOUSE KEY<>1 : WEND : z=ZONE(0) : WEND
22180 IF z<28 THEN CHANGE MOUSE 3+z : WHILE MOUSE KEY<>0 : WEND : GOSUB 22260
22190 IF z=28 THEN done=1
22200 IF z=29 THEN n$=""
      " : LOCATE 14,10 : PRINT n$
22210 WEND
22220 FOR a=1 TO 13 : IF MID$(n$,a,1)="" THEN MID$(n$,a,1)=" "
22230 NEXT a
22240 IF RIGHTS$(n$,1)="" THEN n$=LEFT$(n$,LEN(n$)-1) : GOTO 22240
22250 RETURN
22260 IF z=27 THEN c$="" ELSE c$=CHR$(64+z)
22270 WHILE MOUSE KEY=0 : WEND
22280 IF MOUSE KEY=2 THEN CHANGE MOUSE 1 : RETURN
22290 x=X MOUSE : y=Y MOUSE
22300 IF x<110 OR x>210 OR y<157 OR y>172 THEN 22260
22310 c=(x-100)/8 : MID$(n$,c,1)=c$ : LOCATE 14,10 : PRINT n$
22320 GOTO 22270
```

Whenever you click on a letter, the mouse pointer becomes that letter, which can be placed anywhere within the name. Line 22300 makes sure the mouse/letter is within range of the name grid and line 22310 calculates exactly where in the name the new letter should appear.

The final program for this month only allows the player to enter his or her initials. This technique was widely used on arcade machines and many computer games back in the early '80s, but has now become quite rare. After examining the code you should

➤ be able to expand it to take 10 or more characters:

```

22000 REM Program IV
22010 MODE 0 : KEY OFF
22020 CURS OFF : HIDE
22030 FOR a=1 TO 3
22040 BOX a*30+80,70 TO a*30+100,90
22050 NEXT
22060 n$=""
22070 FOR a=1 TO 3
22080 sp=1 : done=0 : WHILE done=0
22090 SPRITE 1,a*30+88,75,sp
22100 IF JOY=4 THEN DEC sp ELSE IF
JOY=8 THEN INC sp
22110 IF JOY=0 THEN i$=INKEY$ : s=
SCANCODE
22120 IF s=75 THEN DEC sp : CLEAR KEY
: ELSE IF s=77 THEN INC sp : CLEAR KEY
22130 IF sp<1 THEN sp=27 ELSE IF sp>27
THEN sp=1
22140 IF FIRE=-1 OR s=57 THEN done=1 :
PUT SPRITE 1 : WAIT VBL.: IF sp=27 THE
N n$=n$+"." ELSE n$=n$+CHR$(64+sp)
22150 WAIT 5 : WEND
22160 NEXT : SPRITE OFF : CLW : RETURN

```

To enter a character either use the joystick or left/right cursor keys to scroll through the available letters. When you find the one you want, press the fire button or the spacebar.

● In the June issue we'll be taking a look at cheat modes. If you have any problems, tips or suggestions for this column, write to me and I'll include the most interesting.

High Score Table

Atari	ST	User		10000
Atari	ST	User		9000
Atari	ST	User		8000
Atari	ST	User		7000
		Phil		6543
Atari	ST	User		6000
Atari	ST	User		5000
Atari	ST	User		4000
Atari	ST	User		3000
Atari	ST	User		2000

Testing the high score table with a dummy value

ROMANTIC ROBOT present

The simplest and best advertisement we ever had:

To cut our overheads, we must sell 6 MONTHS STOCK in 6 WEEKS! Thus

**You have an absolutely unique chance to buy a
MULTIFACE ST at HALF-PRICE!**

There was NEVER such a fantastic opportunity as NOW:

simply buy a MULTIFACE directly from us

between 1st April - 15th May and pay NOT £59.95, but

★£29.95 ONLY★! Multiface - your ST 1ST choice.

Send SAE for full Multiface details. Kindly note, that the MULTIFACE is not compatible with STE.

MULTIFACE must not be used to copy, reproduce or infringe in any way any copyright material without the clear permission of the copyright owner. ROMANTIC ROBOT neither condones nor authorises the use of the MULTIFACE ST for the reproduction of copyright material - to do so is illegal.

I order a Multiface (incl. Disk Organizer) at £29.95 ★ plus P&P ★ UK £1, Europe £2, Overseas £3

I enclose a cheque/PO/cash for £..... or debit my Access/Visa No. Card Exp.....

Name/address.....

ROMANTIC ROBOT UK LTD 54 Deanscroft Ave, London NW9 8EN

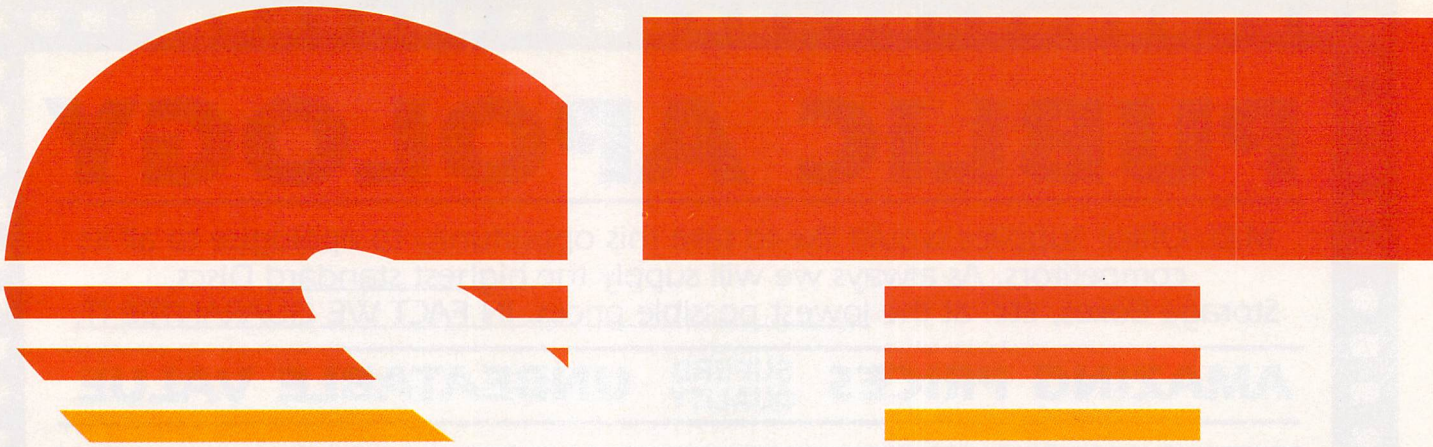


24 hrs



01-200 8870

stu



A moving experience

WE have all seen amazing arcade games containing hordes of sprites, all moving independantly of each other or following a leader. No matter what the final appearance, you can rest assured that the programmer has forced them to follow pre-defined paths, which explains why certain aliens always appear at exactly the same point in a game.

For the next couple of months I'll be showing how to implement this feature in our own games, starting with simple examples and working up to some very complex techniques.

There are two ways of making a sprite travel around the screen. The first and probably most often used is by using the MOVE command, but with many large sprites onscreen this can become too slow.

The second way is to store each coordinate a sprite is to appear at in an array and use the SPRITE command to display it at each one in turn. Don't worry if this sounds complicated, as we'll be examining this

Phil Lawson cast his eye on the humble sprite and comes up with some simple but interesting effects

method in detail next month.

The simplest way to control a sprite is to make it travel from one side of the screen to the other and back again in a continuous loop. Boot up STOS, then load Program I from the Cover Disc and run it.

You should recognise lines 26000 to 26030 from previous articles – they set the colours to those stored in the sprite bank. Each program this month will use this routine. Line 25050 tells the sprite that it will be moving 200 pixels to the left and then 200 to the right.

The letter L means that when the movement is finished the sprite should loop back to the beginning of the

MOVE command. Just because the screen is 320 by 200 pixels in size, there's no reason why our sprites cannot move outside this area. Program II shows a sprite appearing from the top of the screen, moving down to the middle of the screen, going left and finally disappearing off the top of the screen. Load it and run it.

Everything in this program is controlled through the two MOVE commands in lines 25040 and 25045. To recap, the syntax of this instruction is:

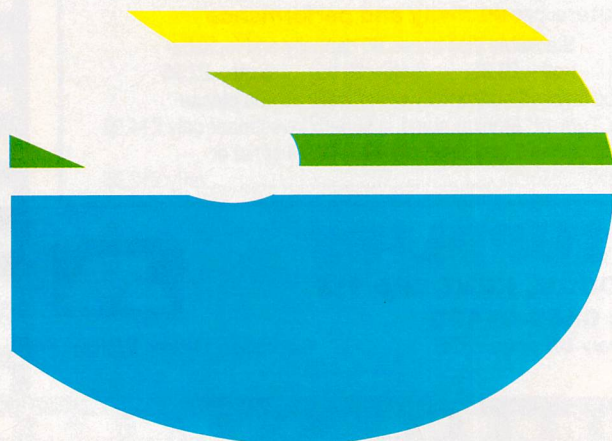
`MOVE speed,direction,steps`

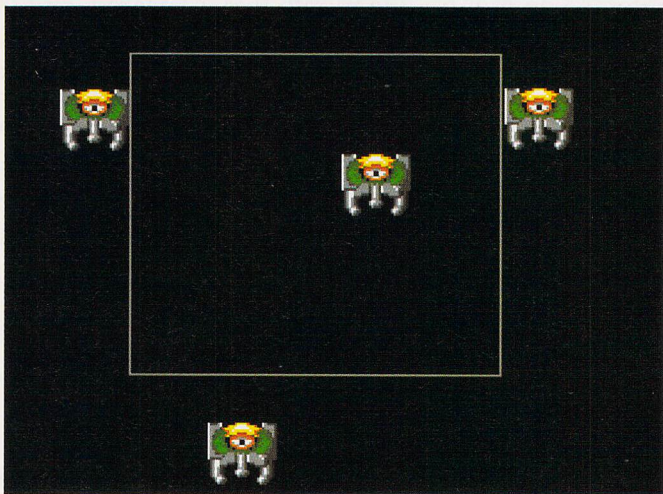
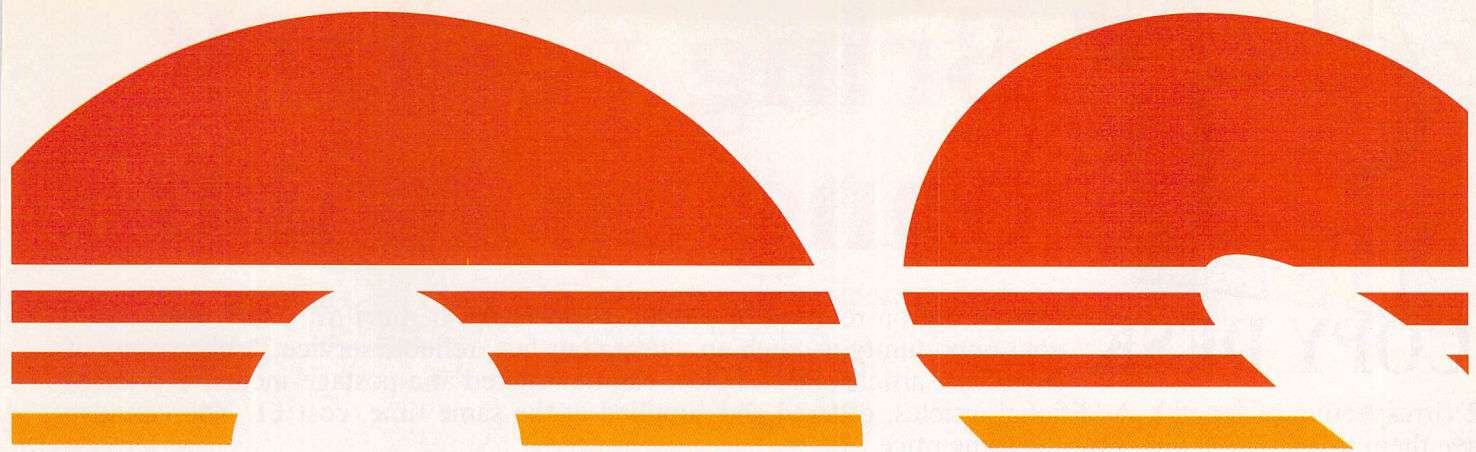
The important parameter is *direction*, which specifies the number of pixels the sprite will move each step. If this number is negative the sprite will move either left or up, where positive values move the sprite right or down, depending whether the command is controlling the X or Y plane.

If the value of *direction* is zero, the sprite will not move in that plane at all. Taking a look at lines 25040 and 25045 shows that the sprite will move down the screen for 100 steps, without moving horizontally at all because the value for X movement is zero. After 100 steps the sprite moves to the left (note the negative value of *direction*). Our sprite will now move up the screen and vanish off the top.

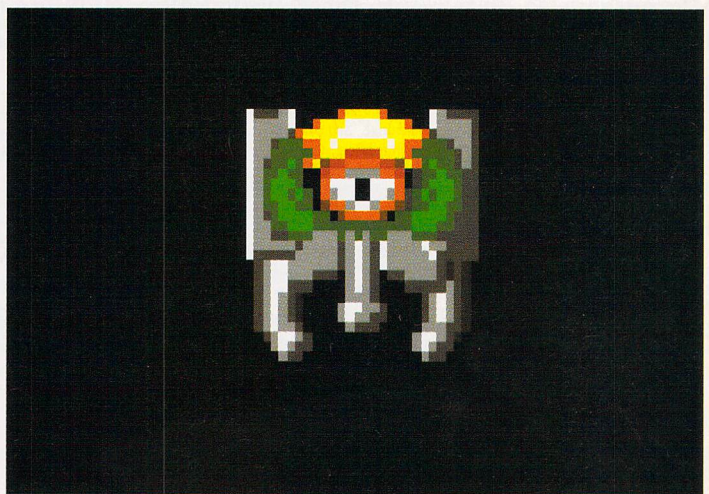
• Using combinations of MOVE X and MOVE Y commands like this can produce very impressive results, including diagonal movement as in Program III.

Sometimes it may be necessary to keep a sprite within a certain predefined area on the screen. As we have seen, we can easily define the path for





One sprite is confined by the box, the others patrol the perimeter



Close up of a sprite. Replace it with your own design if you like

our sprite to follow with the MOVE commands, but unfortunately this doesn't allow for random movement. The best way around this is to set maximum and minimum values for the X and Y sprite coordinates and keep checking that it doesn't go past them.

Two other variables are used to show the sprite's direction in both the X and Y axis, and whenever the sprite reaches the edge of its area the appropriate variable is altered. Load and run Program IV to see this working.

The variables *xp* and *yp* are the initial position of the sprite chosen at random, and *xd* and *yd* are the directions the sprite will travel. When *xd* is one the sprite moves to the right, and a value of minus one moves it to the left. If the value is zero our sprite won't

move at all in the X direction. The same goes for *yd*, except the movement is vertical.

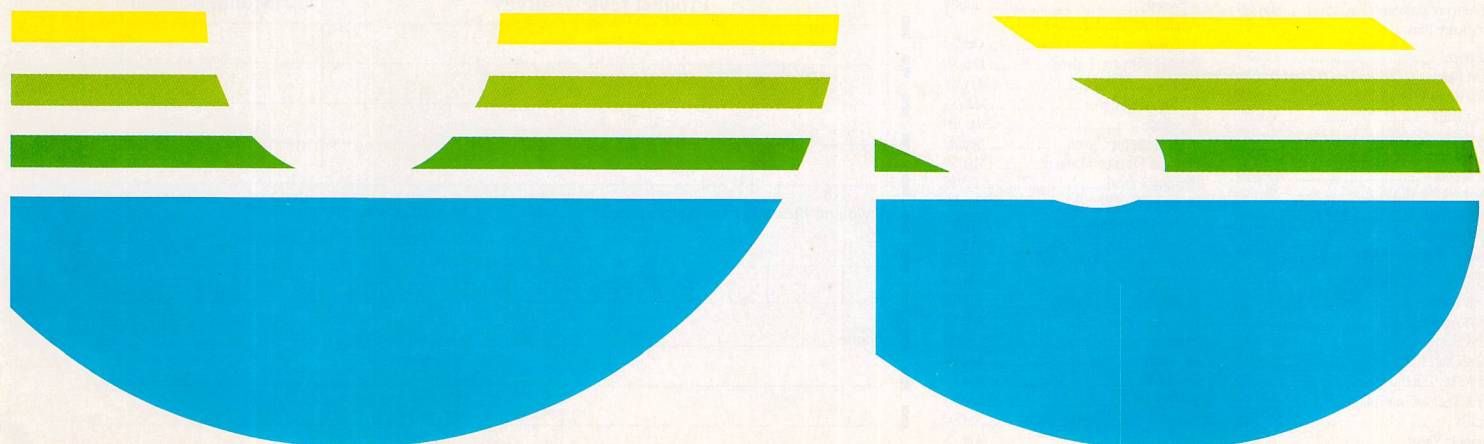
In Program IV, after every 300 counts – or six seconds – the sprite changes direction and to make sure the new directions are different from the last ones, two variables are used to remember the last values. These are *oxd* and *oyd*. Line 25040 just draws a box on the screen so you can be sure the sprite is staying within the area.

The two important lines are 25120 and 25130. This is where the checks are made to ensure the sprite hasn't reached the edge of its allocated area. If it has, the value of *xd* or *yd* is negated. This simply changes any value of one to minus one and vice versa, therefore altering the direction. If the value is zero, it will remain zero.

We'll be covering how several sprites can all follow the same path next month, along with some slightly more complex sprite techniques, but to get you in the mood Program V is a very simple program with two sprites moving around the edge of the screen.

We have covered enough MOVE commands to enable you to discover exactly how this program works. Our final program this month shows four sprites and covers all the techniques we have discovered. I mentioned at the beginning that some sprites can be forced to follow others, and Chase is a very simple program which demonstrates this technique.

● Can you discover how it works? I'll leave you with that little brain teaser until next month, when I'll be looking at this and other STOS sprite movement routines.



RUNTIME

ISSUE TWO

Crammed from cover to electronic cover with programming ideas, help, advice and solved problems, RunTime is the World's first magnetic media magazine totally dedicated to ST programmers...

ON THE ELECTRONIC PAGES...

PROGRAMMER'S CLINIC

If you have an embarrassing silicon problem then come along to the friendliest clinic around. All appointments made in the strictest confidence.

GAMES WRITING

Whether you're an experienced STOSer or new to the game, you'll find a wealth of hints, tips and help for arcade action programmers.

DEGAS DISPLAY

Software is increasing in size faster than disks are dropping in price, save your pocket money by cramming more graphics than ever onto your floppies.

Each month cover disk submissions arrive by the sack load and poor old Postman Pat's (and his black 'n' white cat) back must be under a tremendous strain. The overwhelming majority of programs are superb, but the storage capacity of ST disks is limited to around 800k which means that only a small number of programs make it to the 'USERDISK'.

One of the reasons why programs take up such a lot of disk space is that they require one or more graphic screens. These are stored in either Neochrome or Degas format and occupy over 32k of disk space.

Since each screen is 32k long, requiring just two means that 64k of disk space is eaten up, even before you start writing a line of program code! What's needed is a way of reducing the amount of space the graphics occupy.

A Degas compressed screen is typically around 12k to 16k long - less than half the size of a standard screen. If compressed screens could be substituted for standard ones, you'd half the disk space required.

Saving compressed screens from Degas is easy, loading them into your programs isn't! But now, we present two amazing utilities that will load and display compressed screens. Just think, you can have lots of breath-taking screens featured in your programs without using valuable RAM memory. We've also provided fully commented source code so you can incorporate the subroutines into your own programs and learn how to construct your own routines - aren't we nice?

FILE SELECTORS

This month's Screenfull challenge has been met by a bundle of GEM file selectors. Chop out the code and insert it into your own programs.

GEM PROGRAMMING

Part 2 of the series delves even deeper into the VDI jungle. We reveal the hidden secrets of printing various text styles and sizes on the screen - and much more!

CLIP ART

Clip-art pictures for you to cut out and paste onto your DTP pages, and sound samples that can be included in your next arcade masterpiece.

SCREENFULL

RunTime's Screenfull is an exciting new challenge. Programmers must write a program (or just a subroutine) that will fit on one standard monitor or TV screen. The program can be a game, utility, graphic designer, music and MIDI editor or even a business program, as long as the source code fits in just one screen, the choice is yours.

This month's Screenfull submissions are all supplied by programmer Janice Murray and demonstrate how to call up GEM's file selector to input a filename from within your own programs. It doesn't matter whether you use HiSoft or Fast BASIC, or even dabble in C - we've catered for all three languages.

Give your programs the professional look by including these tailor-made source code subroutines into your own programs. It couldn't be simpler!

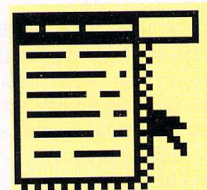
TEST CARD

Don't miss out on the desktop video revolution! These two handy utilities will display your very own custom designed test card. Use them straight from the disk or modify and re-compile the source code to make something completely new. They are also ideal for tuning up duff, out of focus monitors and televisions - but don't go poking your digits into the back of them unless you're auditioning for Death Wish V. RunTime's editor Roland Waddilove, examines the VDI. To see what can be achieved using a few simple VDI functions.

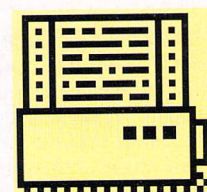
GETTING STARTED

RunTime appears as folder on side one of UserDisk July. To start reading the magazine, simply double-click the RunTime folder, then double-click the magazine icon. After a few seconds, you'll be presented with an opening screen consisting of a title page and standard GEM menu bar.

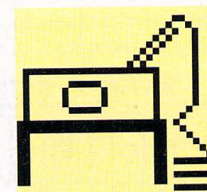
All the articles, features, news and reviews appear under the menus. For example, there might be a feature about accessing upper memory using C. To read this, select the correct menu from the bar, pull down the highlighted bar and release on the correct feature. Soon, you'll see a new screen consisting of the first page of the feature and a row of icons (show below) at the foot of the screen. You can now begin reading. If you wish to return to the menu, print a feature and so on, use these icons...



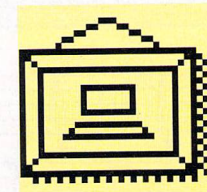
This is the icon which enables you to return to the opening GEM menu bar. Click this if you want to stop reading the current feature



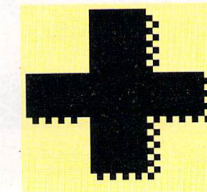
Use this icon if there's a particular page of the feature which interests you and which you wish to have on paper



If you need to print the entire feature, tutorial and so on, this is the icon you need. Click here to send the complete feature to your printer



Often, as you're reading a feature, there'll be an associated picture which helps to illustrate a point. Click this icon to see the picture



Everyone, even learned programmers need help occasionally, if you find that some friendly advice is needed, click the cross

RUNTIME

ISSUE THREE

The world's first totally dedicated electronic media magazine for all ST users goes from strength to silicon-based strength . . .

ON THE ELECTRONIC PAGES...

PROGRAMMER'S CLINIC

Got a problem? Then come along to the friendliest clinic around. If Mathew Lodge can't solve it he'll post it on the electronic noticeboard and invite our talented readers to help out. Solutions will flood in before you can say arrghh...

ARCADE GAMES WRITING

Struggling with STOS? Phil Lawson provides a helping hand. Dissecting the great granddaddy of all arcade games Breakout, he shows how not to lose your marbles!

EXPLORING GEM

The ST's virtual device interface is a minefield for the unwary - a wrong call, a parameter missing, and boom! The computer prints those cute little bombs all over the screen. Roland Waddilove comes to the rescue and provides a safe route.

SCREENFULL

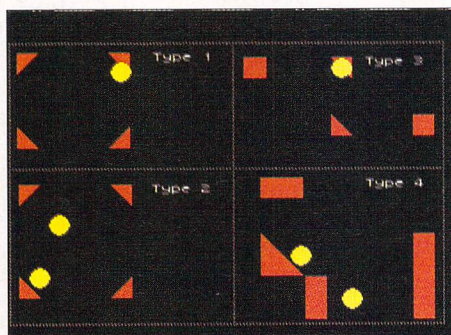
The ultimate challenge for programmers is to write a useful routine in just one screen of code. It can be done, and here's another microscopic marvel from our talented readers.

The submissions for RunTime keep on rolling in and the Post Office has laid on a special van just to deliver our sack-load of mail each week. We cram as much into RunTime as possible - in fact, even more because we're using advanced compaction techniques that squeeze over a megabyte of programs, utilities and articles onto one double sided disk. And as if that's not enough, it can be read by single sided drives, too!

In Mathew Lodge's disk-based clinic this month there are questions on C, the STE as well as a guide to using GDOS fonts. There's a plea for someone to explain what the STE's Cookie Jar feature is, a reader seeking assistance with Gulam (gosh!), and everything you ever wanted to know about GDOS fonts but were afraid to ask.

Screenfull is an exciting new challenge for programmers - to write a program (or even just a subroutine) that will fit on one standard monitor or TV screen. That's about 22 lines. The program can be a game, utility, graphic designer, music and MIDI editor or even a business program, the choice is yours. You can also use any programming language such as Basic, C, Pascal, STOS or 68000 machine code but remember, the whole of the source code must fit onto one screen.

This month's challenge has been answered by programmer Sean Cleaver of East Hoathly in Sussex. It's a password program written in 68000 assembly language using Devpac 2. Assemble it and save it in the AUTO folder of your boot disk. When the ST is booted the program prompts you for a password (458, but



it can be changed to any numbers or digits), and won't continue unless you enter the correct one.

If you are burning the midnight oil trying desperately to create a block-buster of an arcade game then Phil Lawson provides some light relief with his continuing tutorial on all aspects of this superb game creator. This month's column is a load of old balls - he demonstrates how to make balls roll around the screen and bounce off objects at the correct angles. Here's a use at last for those dull laws of motion devised by that awfully clever Newton chappie you learnt about at school!

For the programming elite who wouldn't be seen dead with a BASIC interpreter in their micro Roland Waddilove continues his exploration of the ST's virtual device interface for C programmers. Remember - Drawing lines, circles and rectangles is easy when you do it Roly's way!

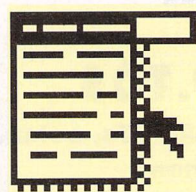
GETTING STARTED

RunTime is a disk-based magazine program with articles providing programming help, hints for games writers and much more.

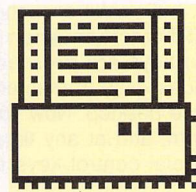
The magazine appears on the disk as a program called MAGAZINE.PRGM, all you need to do to read it is double click on the program icon. You'll see a GEM menu bar at the top of the screen, moving your mouse pointer over it reveals several drop down menus.

These menus contain the titles of the programs, articles, games and utilities on the cover disk. Click on a menu option and the article will be loaded and displayed on the screen. If your TV isn't very clear you can print out the article, then study it at your leisure.

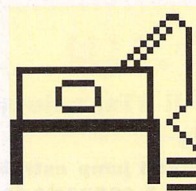
At the foot of the screen you'll see several icons. Here's what the main ones do when you click on them with the mouse...



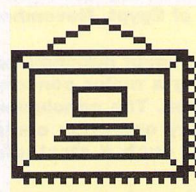
This is the icon which enables you to return to the opening GEM menu bar. Click this if you want to stop reading the current feature



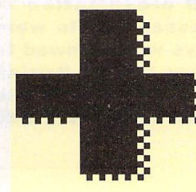
Use this icon if there's a particular page of the feature which interests you and which you wish to have on paper



If you need to print the entire feature, tutorial and so on, this is the icon you need. Click here to send the complete feature to your printer



Often, as you're reading a feature, there'll be an associated picture which helps to illustrate a point. Click this icon to see the picture



Everyone, even learned programmers need help occasionally, if you find that some friendly advice is needed, click the cross



RUNTIME

ISSUE FOUR

The world's first totally dedicated electronic media magazine for all ST users goes from strength to silicon-based strength . . .

ON THE ELECTRONIC PAGES...

PROGRAMMERS' CLINIC

Questions and answers about Spectrum 512 pictures, bombing programs, and changing dialog button text make Mathew Lodge's clinic this month. Setting the double click speed from the AUTO folder also makes a come back. Got a problem? The Clinic will soon solve it.

GEM TUTORIAL

Get to grips with the ST's graphic environment manager and harness the power of the best micro available. The virtual device interface is the key to all graphic output and Roland Waddilove's step-by-step guide, will lead you gently by the hand through this programming minefield.

Each month Atari ST User's disk-based programming magazine becomes more popular. The cover disk submissions are rolling in thick and fast and although the vast majority of programs are stunning, we're stuck with the ST's minuscule 800k floppy disks - only the lucky few make it to the duplicators. Don't panic if your submission hasn't yet been used - you're probably on the waiting list!

In this month's packed Programmer's Clinic there's a mixed bunch of problems and solutions. Maybe yours is answered this month! First up, is the thorny problem of setting the mouse double click speed in the AUTO folder to improve mouse response. Normally, this can't be done because GEM hasn't been initialised. One clever Clinic reader however, has used a bit of lateral thinking and worked around the problem.

Les Kneeling has also replied to the query from Ian McCall about changing the text in a dialog box in Laser C. Les provides a full worked example that can be compiled and run.

MIDGET GEM

RunTime's GEM programming tutorial continues its exploration and documentation of the ST's almost impenetrable virtual device interface, or VDI for short. This is the part of GEM responsible for drawing lines, arcs, circles, boxes and filling shapes.

This month, Roland Waddilove sheds some light on rectangles and fill patterns. Faced with a choice of 24 fills and 12 hatch patterns, filled and hollow rectangles, borders and rounded corners, the article is packed with facts, figures and programming examples.

Screenfull is an exciting challenge for programmers - to write a program (or just a subroutine) that will fit on one standard monitor or TV screen. That's about 22 lines.

This month's two Screenfull submissions are supplied by talented programmer Les Kneeling.

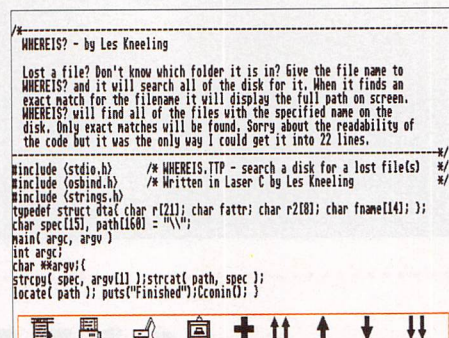
SCREENFULL

Two handy utilities have been supplied in response to this month's Screenfull challenge, TAB22, a program which converts tabs to spaces and Whereis? which searches for lost files on your hard disk. Could you write these in only 22 lines of code? See how it is done...

ARCADE PROGRAMMING

Phil Lawson takes a break from discussing STOS programming techniques, to give you the bare bones of a fascinating puzzle game. The aim is to drop balls into a vertical maze so that they fill the holes at the bottom. It's up to you to finish it off, so boot up STOS and get cracking!

The first is the classic utility program Whereis? which tracks down an elusive file that you can't find. When you have 30 or 40Mb of hard disk space and possibly several hundred files, finding the one you want can be difficult. Pass the filename to WHEREIS?.TTP and it will seek it out and tell you the complete path name.



TAB22 removes tab control codes from program source code and replaces them with spaces. Of course it would not be sufficient to simply replace each occurrence of a tab character with say four spaces. This would result in a ragged listing. What's actually needed, is to replace the character with the required number of spaces to move the cursor to the next tab column. And that's the task TAB22 performs.

Over the past couple of months, STOS games programming expert Phil Lawson has been showing how to bounce sprites all around the screen and in particular, off various obstacles.

You may have been wondering where all these techniques and methods were leading, in which case your question is answered this month. What Phil's done, is to write the main controlling loop of a strategy game, but it's up to you to finish it off. This should put your programming skills to the test.

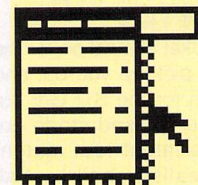
GETTING STARTED

RunTime is a disk-based magazine program with articles providing programming help, hints for games writers and much more.

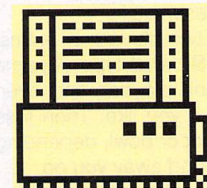
The magazine appears on the disk as a program called MAGAZINE.PRQ, all you need to do to read it is double click on the program icon. You'll see a GEM menu bar at the top of the screen, moving your mouse pointer over it reveals several drop down menus.

These menus contain the titles of the programs, articles, games and utilities on the cover disk. Click on a menu option and the article will be loaded and displayed on the screen. If your TV isn't very clear you can print out the article, then study it at your leisure.

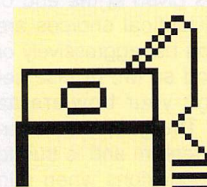
At the foot of the screen you'll see several icons. Here's what the main ones do when you click on them with the mouse...



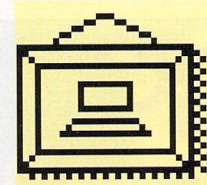
This is the icon which enables you to return to the opening GEM menu bar. Click this if you want to stop reading the current feature



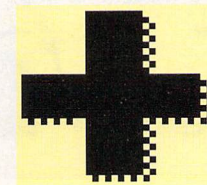
Use this icon if there's a particular page of the feature which interests you and which you wish to have on paper



If you need to print the entire feature, tutorial and so on, this is the icon you need. Click here to send the complete feature to your printer



Often, as you're reading a feature, there'll be an associated picture which helps to illustrate a point. Click this icon to see the picture



Everyone, even learned programmers need help occasionally, if you find that some friendly advice is needed, click the cross

RUNTIME

ISSUE FIVE

Once again, the world's only dedicated ST programming magazine on disk is yours to have and to hold...

ON THE ELECTRONIC PAGES...

PROGRAMMERS' CLINIC

Mathew Lodge deals with the confused issue handles for GEMDOS from C languages running under the UNIX work-alike shell Gulam and explains how handles are opened to allow the ST to deal with programming events.

In the first of a new occasional series, Mathew looks at all the different versions of TOS that Atari have released over the years, from 1.0 right up to the brand new TOS 3.0 in the new TT machines (see the news pages). This month he examines the improved TOS 1.4 file selector and explains how to get the most out of it, including calling its new features such as writing text to the title line.

GEM TUTORIAL

In the past four issues of RunTime, Roland Waddilove has guided both novices and experienced programmers through the ST's Virtual Device Interface. The VDI allows programmers to access the vast power of the ST quickly and easily with a few simple calls to GEMDOS. In the last of the series, Roland documents the GEMDOS VDI calls which enable programmers to draw circles and ellipses from within a GEM environment.

SCREENFULL

The popular Screenfull Challenge continues this month with two new reader submissions, in GFA Basic and STOS Basic.

C. Warren of Bury St. Edmunds has provided a neat little accessory to control the Atari blitter chips now fitted as standard in the STes and Megs. Available from any program that gives you access to the menu bar, you can turn the blitter on and off whenever you need it. Both the source code and the compiled accessory, which are written in GFA Basic, are included on the disk.

The second screen this month comes from Paul J. Dixon and is written in STOS Basic. In this simple arcade-style game you guide your space ship through an obstacle course created by a hostile alien race.

Unfortunately, your ship has been damaged and can only move up and down! The only way to change direction is by cannoning off something else on the screen, but doing so will deplete your limited shields.

NB. This program is designed to be run from within STOS Basic only, and a compiled version has not been included on the disk.

ARCADE PROGRAMMING

Now that Atari has started bundling Mandarin's excellent Basic-like programming language STOS with many new STs and STEs there are an increasing number of readers who want to explore this powerful and versatile language to write their own programs.

For the benefit of both new and experienced users of STOS, Phil Lawson continues his tutorial this month with a program to allow the paths of on-screen sprites to be easily defined and stored in an array. This program has been compiled to a .PRG file which can be clicked on from the desktop. To save space on the disk, the file has also been packed and will automatically de-pack itself when it's run, so be prepared for a short wait.

The advantage of storing sprite paths in arrays is that they take less space in memory and are easy to manipulate to create new and exciting paths for your mutant aliens, giant insects or whatever.

Also included on the disk are details of converting this program to run under different dialects of Basic, or even assembler, so this program is available to you no matter what language you use to create your masterpiece.

PREPARE TO BE AMAZED!

Over recent months we've received a lot of mail from readers suggesting ways to improve the magazine, and more often, to improve the Cover Disk. Well, never let it be said that we don't listen to you, because from next month, you'll notice a change in our Cover Disk.

It seems that we were pitching a bit high. Too many readers were having difficulty loading the programs and using the Disk. So to avoid further frustration and tearing out of hair, clear and concise instructions will be published next issue to allow the complete ST novice to get the maximum benefit from the 1Mb of programs we cram on the disk. Then, in

following issues, we'll include a brief, but helpful reminder so that no-one gets lost on their way into that huge vault of priceless information, the Disk.

RunTime is in for a hit too. It will include more features than before, so you can read up on what you're about to do with the programs included on the Disk. Best of all, there will be a new programming feature, aimed at helping the novice programmer get started. For once, this will be a truly reasonable and simple introduction to programming. We won't go from A to X,Y and Z without filling you in on what comes in between. Honest.

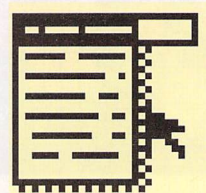
GETTING STARTED

RunTime is a disk-based magazine program with articles providing programming help, hints for games writers and much more.

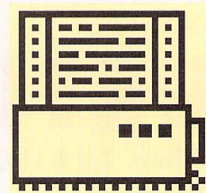
The magazine appears on the disk as a program called MAGAZINE.PRG, all you need to do to read it is double click on the program icon. You'll see a GEM menu bar at the top of the screen, moving your mouse pointer over it reveals several drop down menus.

These menus contain the titles of the programs, articles, games and utilities on the cover disk. Click on a menu option and the article will be loaded and displayed on the screen. If your TV isn't very clear you can print out the article, then study it at your leisure.

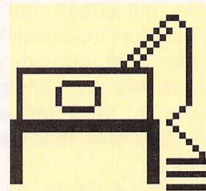
At the foot of the screen you'll see several icons. Here's what the main ones do when you click on them with the mouse...



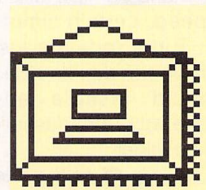
This is the icon which enables you to return to the opening GEM menu bar. Click this if you want to stop reading the current feature



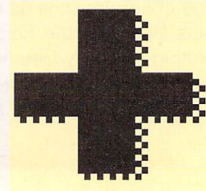
Use this icon if there's a particular page of the feature which interests you and which you wish to have on paper



If you need to print the entire feature, tutorial and so on, this is the icon you need. Click here to send the complete feature to your printer



Often, as you're reading a feature, there'll be an associated picture which helps to illustrate a point. Click this icon to see the picture



Everyone, even learned programmers need help occasionally, if you find that some friendly advice is needed, click the cross

RUNTIME 6

The world's first totally dedicated electronic media magazine for all ST users goes from strength to silicon-based strength

ACCSEL

With machines such as the ST, drop down Desk Accessories are a great idea because the machine will only run one application at a time. DAs enable you to use functions not normally included in applications. Being able to format a disk from within a word processor can be a life-saver when it's time to save your work and you've run out of disk space.

The ST has only six slots for DAs, and it soon becomes apparent that this is not enough. One solution is multi-function DAs such as Harlekin or Goodies, but these tend to take up more memory than most users can afford. Programs such as Multidesk allow more than six DAs to be loaded at once, but eat memory at a phenomenal rate.

Most people need different DA setups depending on which program they're using. Selecting the right DAs to boot the machine is a difficult process involving the Desktop Show Info. option to rename un-needed DAs to .ACX, so they won't be loaded.

If you have a big program which requires all the memory available on your ST, then the DAs have to be disabled.

But now, Les Kneeling has come to the rescue of all those ST users struggling to get their DAs set up just the way they want them. His small AUTO folder program presents you with a list of all the accessories on your boot disk, then lets you select which of them to load. An end to those 'Out of Memory' errors or having the wrong DAs loaded.

As well as the compiled ACCSEL program, this month's RunTime includes an article by Les on writing AUTO folder programs and the source code for ACCSEL, so you can see how it's done.

MIDI FILES

Perhaps one of the best parts of the ST's design is the inclusion of the standard MIDI In and Out ports. The ST has accumulated a wealth of excellent music programs, which have made it the first choice for both professional and amateur musicians.

Applications such as Cubase, the Steinberg Pro 24 Sequencer, and Band in a Box, have given the ST a reputation second to none in the music arena.

All of these programs use the standard MIDI language to communicate with equipment such as sound modules, keyboards and sequencers.

MIDI programs have evolved their own set

HARD CASH FOR YOUR SUBMISSION

Have you programmed a stunning new game, a utility no ST user should be without or a fabulous demo?

Then we want to hear from you!

Here at Atari ST User, we're always keen to see your programs included on our CoverDisk, and we might even give you money for them! GASP!

Unlike some magazines we don't promise overnight fame and immeasurable wealth, but a program included on the Userdisk will be seen and used by an enormous number of ST owners - who knows what that will lead to?

If you have a program that you think is up to our high standards, please send it in for us to look at. Send your program on disk, together with a README.DOC file giving a brief outline of the pro-

gram, what it does and how it works.

Full instructions must also be included, since we can't publish a program that doesn't include clear instructions for our readers.

If possible, we would appreciate a printout of the document files. Remember to write your name and address clearly on the disk label so we can identify your disk in this chaos we call an office!

If you want the disk returned, please remember to include a stamped, addressed envelope. If you don't, the disk will eventually be dragged to that great Trashcan in the sky.

So, send your disks to: Simon Lawson, Userdisk Submissions, Atari ST User, Europa House, Adlington Park, Macclesfield SK10 4NP

of standard file formats for exchanging data, and in the first of a two part series for RunTime, Paul Andreas Overaa examines the way MIDI data is stored, and provides hints and tips to help you write programs to access MIDI data files.

STOS TUTORIAL

If there is anyone out there who's never heard of Dungeon Master, where have you been living for the past two years, Mars?

Dungeon Master took the ST games world by storm, and had hordes of gamers crouched over their machines until the wee small hours of the morning, desperately searching for that last elusive key that would give them access to the Firestaff and final victory over Lord Chaos.

One of the big selling points for this type of game is the 'first person perspective' that enables the players to view the game world through the eyes of their character. Dungeon Master, Galderon's Domain and Bloodwych, have made this type of game popular and programmers have begun to experiment with creating their own games.

This month, Phil Lawson continues his tutorial and explains how to create a 3D game using STOS arrays and variables. For the first time, a Dungeon Master-style game is within reach of the average programmer.

PROGRAMMERS CLINIC

Programming your ST, whether from BASIC or Assembler can be very rewarding. There's nothing quite like compiling a program you've spent hours working on and seeing it up and running for the first time.

If your program's good, it might end up on our Userdisk where thousands of people will see and use it. Public domain libraries abound with excellent programs written by people in their spare time.

But what happens if your labour of love doesn't work, or some rogue instruction is running amok in the GEM VDI routines? Maybe your machine decides not to work or a stack pointer points to the wrong place and bombs scatter the screen? Worry not, for Mathew Lodge is here.

Every month Matt dives into the programming complexities of the ST's internals and sorts out perplexing problems, baffling bombs and all those other niggling little wrinkles that happen to your ST just when you're least expecting it.

cover
1990
58 disk
december

RUNTIME

8

Programming from beginning to end, from those in the know. It's all here on the world's first dedicated electronic media magazine for the ST...

GETTING STARTED

RunTime is a disk-based magazine program with articles providing programming help and hints for games writers and much more.

The magazine appears on the disk as a program called MAGAZINE PRG and all you need to do to read the magazine is double click on the program icon. You'll see a GEM menu bar at the top of the screen – moving your mouse pointer over it reveals several drop-down menus.

These menus contain the titles of all the programs, articles, games and utilities on the cover disk. Click on a menu option and the article will be loaded and displayed on the screen. If your monitor isn't very clear you can print out the article and study it at your leisure. At the foot of the screen you'll see several icons. Here's what the main ones do when you click on the with the mouse...



This is the icon which enables you to return to the opening GEM menu bar. Click this if you want to stop reading the current feature.



Use this icon if there is a particular page of the feature which interests you and which you wish to have on paper.



If you need to print the entire feature, tutorial and so on, this is the icon you need. Click here to send the complete feature to your printer.



Often, as you're reading a feature there'll be an associated picture to help illustrate a point. Click on this icon to see the picture.



Everyone, even learned programmers, needs help occasionally. If you find that some friendly advice is needed click the cross.

ON THE ELECTRONIC PAGES

STOS PROGRAMMING

STOS, the Games Creator from Mandarin Software is probably one of the most popular implementations of BASIC available for the ST. Quick, easy to use and specially designed for programming bright and colourful games, STOS has become a firm favourite with users.

In his article in the October Runtime magazine, Phil Lawson introduced STOS programmers to the concept of defining a sprite path as a data statement.

This month Phil goes further, explaining how to load and manipulate the sprite path data within programs. An updated version of the Sprite Path Designer program is also included, so don't worry if you missed it the first time around.

FIRST STEPS IN BASIC

BASIC is often the very first programming language computer users try. It's bundled with new ST, it's simple and easy to learn, but at the same time it provides the power to access the hardware which makes the ST special.

This month Peter Phillips begins his tutorial on the ins and outs of BASIC programming.

The Print statement lets BASIC communicate with the outside world, and there are several tips for getting the most out of it.

You can follow this BASIC programming series using the demo version of PowerBASIC from HiSoft which is included on the CoverDisk this month. Remember to take a look at the full instructions on the Disk, in Runtime.

GFA RESOURCE PROGRAMMING

GFA BASIC is a fast and powerful implementation of the BASIC programming language that is rightly popular among power users. Both version 2 and 3 are still best sellers and new users are being introduced to the language all the time.

For programmers who've gone beyond the beginners stage and want to branch out a bit, John Peters begins a new series on how to program the powerful features of GEM directly from GFA BASIC.

You can learn how to program GEM alert boxes, dialogs and menus in order to make your program easy to use and give it a really professional look. Programming GEM isn't hard, and our tutorial makes it an even simpler process.

DISKS, VIRUSES AND IMMUNISATION

Following the Green Goblin Virus on the December CoverDisk the ST User team have taken steps to ensure that such an unwelcome visitor never again creeps onto our Disk.

As from last month we are using a special boot sector on the CoverDisk.

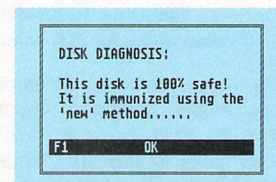
Created and programmed by Richard Karsmaker, the author of the CRL Virus Killer, this clever piece of code sits on the boot sector of a floppy disk and prevents viruses from infecting the disk. This immunisation code

creates an executable boot sector on the disk. Most of the Virus Killers available commercially such as AVK recognise this code and report that the disk is immune to virus infection.

Other less sophisticated programs simply report that the boot sector of the disk is executable. Don't worry about it as this is exactly as it should be and shows the immunisation code is on the disk and doing its job.

Remember, no matter how safe you may think your disks are, never relax your anti-virus

precautions. Check all new disks with a Virus Killer and try to keep your disk write-protected whenever you can.



Richard Karsmakers AVK program from CRL. This is the February CoverDisk under examination

RUNTIME



Programming from beginning to end, from those in the know. It's all here on the world's first dedicated electronic media magazine for ST...

3D GRAPHICS SERIES

Flight simulators pioneered the use of 3D graphics to create realistic landscapes easily and quickly. Sims like F-19 and ProFlight rely on filled polygon landscapes to make them fast and visually exciting.

This month in 'USER Andrew Tyler begins a series of articles on how to program 3D graphics on the ST. To go with these articles, Runtime will include example programs and

GETTING STARTED

RunTime is a disk-based magazine program with articles providing programming help, hints for games writers and much, much more.

The magazine appears on the disk as a program called MAGAZINE.PRG and all you need to do to read it is double click on the program icon. You'll see a GEM menu bar at the top of the screen. Moving your mouse pointer over it reveals several drop-down menus.

These menus contain the titles of all the programs, articles, games and utilities on the CoverDisk. Click on a menu option and the article will be loaded and displayed on the screen. If your monitor isn't very clear you can print out the article and study it at your leisure.

At the foot of the screen you will find several icons. Here's what the main ones do when you click on them with the mouse...



This is the icon which enables you to return to the opening GEM menu bar. Click this if you want to stop reading the current feature.



Use this icon if there is a particular page of the feature which interests you and which you wish to have on paper.



If you need to print the entire feature, tutorial and so on, this is the icon you need. Click here to send the complete feature to your printer.



Often, as you're reading a feature there'll be an associated picture to help illustrate a point. Click on this icon to see the picture.



Everyone, even learned programmers, needs help occasionally. If you find that some friendly advice is needed click the cross.

machine code files to enable you to experiment with 3D graphics on your own.

The folder 3DGRAPH contains all the files which are needed to assemble the program 3D_01.PRG using the Devpac 1 Assembler which is included on the disk. These files have been written to work with both Devpac1 and Devpac2.

Anyone with another assembler should be able to get the program to compile with only minimal modifications. See the Runtime Magazine on the disk for a detailed list of all the source code files and how to use them.

GFA RESOURCE PROGRAMMING

GFA BASIC is a fast and powerful implementation of the BASIC programming language. Available in several versions, it continues to be one of the best selling languages available on the ST.

The ST has a friendly GEM interface to shield the user from the complexities of programs. Too many programs don't use GEM because people regard it as being too difficult to implement.

Programming GEM from GFA BASIC isn't hard, and our series makes it even easier. You'll find that it'll take very little extra effort using the GEM interface to give your program a professional look and feel.

Accessing GEM dialogs and alerts from GFA BASIC is easy. John Peters continues his series for more advanced GFA programmers by explaining how to create radio buttons. Using what has been learnt so far, he creates a simple program using GEM to control a printer.

FIRST STEPS IN BASIC

Over the years, BASIC has proved to be an excellent language with which to learn to program with. There are many dialects and

versions available for almost every computer. It's fast and flexible, and BASIC has gained a massive following among both professional and occasional users.

Well, all of this is fine if you're an experienced and confident user, but what happens if you're just starting out on your ST programming career?

Never fear, ST User is here with First Steps in BASIC. Peter Phillips will lead you gently through the various functions and facilities offered by this versatile language. Using PowerBASIC, from programming specialists HiSoft, Peter gently introduces the novice user to the world of programming.

In the second part of the tutorial, Peter continues the series by looking at the way BASIC deals with mathematical calculations and expressions. Why should a simple sum like $2+3*2$ have two possible answers? With the aid of an elderly relative, Peter will explain.

STOS TUTORIAL

The STOS programming language from Mandarin Software has proved to be very popular. Quick, easy to use and designed to create bright and colourful games, STOS has become a firm favourite with programmers.

This month Phil Lawson, our STOS expert, shows you how to create a realistic 3D maze. Using full screen scrolling and a maze over nine times the size of the screen, this is a truly stunning game.

Make your way through this world, picking up the gold pieces and avoiding the marauding baddies. Locked doors bar your way, so you've got to find the right key for each door which is not easy as you can only carry one key at a time.

Once you've explored the game, you can follow the article and complete any changes you want to make!

YOUR DISK NEEDS YOU

At 'USER we're always on the look out for new programming talent. We know that ST users the world over love programming on their machines, whether they use BASIC or machine code. Games, demos, utilities and desk accessories, we're interested in 'em all. 'USER will pay for any program a reader sends in that's then used on the disk, so this could be your big chance. Not only will your program be seen and used by thousands of people, you could earn a tidy bit of cash as well.

So, if you've programmed something that you think we should know about, send it in for the CoverDisk. We'll take a long, hard look at it and if it's good enough it will appear on a future CoverDisk.

When you send a program in to us, please make sure that:

1. The program works! If you can, test your program on more than one type of ST.
2. You've included all the relevant documentation files on the disk. If it's possible include a printout.
3. The program is all your own work, and you own the copyright.
4. If you want your disk(s) back, remember to include a stamped addressed envelope. We cannot return disks without one.
5. You've included your name, address and a daytime telephone number.

Send your submissions to: CoverDisk Submissions, Atari ST User Magazine, Europa House, Adlington Park, Macclesfield SK10 4NP.

RUNTIME 10

The definitive lexicon of ST programming from those in the know. It's all here in the world's first dedicated electronic programming magazine for the ST...

Create 3D graphics you would swear were the real thing with the second instalment of our informative graphics series written by University lecturer and enthusiastic ST programmer, Andrew Tyler.

From first principles to extended programming practices, Andrew guides you through graphic construction.

And to complement his series, RunTime features a variety of programs and code

fragments which help to illustrate the concepts which are detailed in the magazine.

Last month, we gave you the popular HiSoft DevPac 1 assembler, this month, there's a collection of code modules which you can use with DevPac to actually practise what Andrew preaches.

We'll put the assembler on the Disk again in a month or two, so if you missed last month's issue, watch out for it.

best of them. We uncover the mysteries of the interface and show you how it's done.

This month, John Peters discusses the intricacies of passing information to editable fields within forms, folder name and file requests and so on.

STOS TUTORIAL

The STOS tutorial continues to be one of our most popular series. In this issue, Phil Lawson, the man who knows the language inside out, continues the maze theme started last month by showing you how to scroll. We also get to meet a few baddies!

ABSTRACT DATA TYPES

This month, Paul Overaa introduces the first of a two part series which lays bare the secrets of Abstract Data Types.

Abstract Data? What's that? The principle of abstraction is to cut away unnecessary information leaving only the pertinent bits of a conversation, C function or whatever, so that you can use the data simply and effectively.

In part one, Paul uses queue descriptor structures and values representing objects' sizes to illustrate the power of the ADT.

FIRST STEPS IN BASIC

Unfortunately, due to a slight technical hitch, this month's First Steps... hasn't made it into RunTime. But don't worry, it hasn't gone for good. Philip Peters will be back next month with the next part of his informative guide.

GETTING STARTED

RunTime is a disk-based magazine program with articles providing programming help, hints for games writers and much, much more.

The magazine appears on side one of the Disk as RUNTIME.PRG and all you need to do to read it is to double-click on it. You'll see a screen with the Atari ST User logo on the left and a list of menu options on the right.

These menus contain the titles of all the programs, articles, games and utilities on the CoverDisk. Click on a menu option and the article will be loaded and displayed on the screen. If your monitor isn't very clear you can print out the article and study it at your leisure.

At the foot of the screen you will find several icons. Here's what the main ones do when you click on them with the mouse...



This is the icon which enables you to return to the opening Main menu. Click this if you want to stop reading the current feature.



Use this icon if there is a particular page of the feature which interests you and which you wish to have on paper.



If you need to print the entire feature, tutorial and so on, this is the icon you need. Click here to send the complete feature to your printer.



Often, as you're reading a feature there'll be an associated picture to help illustrate a point. Click on this icon to see the picture.



Everyone, even learned programmers, needs help occasionally. If you find that some friendly advice is needed click the cross.

GFA RESOURCE PROGRAMMING

What is the single most used feature to make programs look really professional? It's the GEM front-end, right?

And how many programs have you had from the public domain, sometimes even commercial offerings, which use some hideous screen display without a Desktop or menu bar in sight?

Possibly the worst 'mistake' any programmer can make is to neglect the GEM user interface. Although it can be clumsy, even getting in the way for experienced users, GEM is a must for beginners and adds a level of 'professionalism' to any program.

You've probably heard some programmers pouring scorn on GEM, but have you stopped to think that they do so probably because they don't have the ability to program its features for themselves?

With our series and a little application, you'll be programming front-ends with the

YOUR DISK NEEDS YOU

Here at Atari ST User, we're always on the look-out for new programming talent. ST owners the world over love to program their machines and we want to publish the fruits of those programming efforts - many of which rival commercial offerings.

If you've written a program which you think would be interesting to other readers, send it to us. Sure, it's easy to be discouraged, "they'll never publish this..." and you'd be right! We can't publish programs we don't receive, OK? Whatever language you've used, BASIC or assembler, Pascal or C, send it to us. We'll evaluate its potential and if it's good enough for the disk, we'll pay you handsomely to boot!

If you're going to send a program, here's what to do:

- Check that it works and that it's as bug-free as you can get it. May sound obvious, but you know your own code. An inexperienced user might press one wrong key and send the whole thing scampering away for cover.
- Make sure that it's ALL your own work. Don't rip off code from anyone else, then send it with your own name.
- Include full instructions.
- Include your name and address.
- If you want the disk back, include a stamped addressed envelope.
- Include a signed declaration bearing these words "I agree to indemnify Interactive Publishing against all possible litigation arising from the use of my software". If you're under 18, get your parent or guardian to sign it too. If you've already sent a disk in and haven't included a declaration, we'll send a form out to you before we use your program. And once you've sent a disk to us, be patient.

We get lots of submissions and we give each and every one careful consideration. Don't worry too much though if you have sent in something, with an SAE, and haven't heard anything since. Some things just take time and can't be hurried...

Now open up the definitive lexicon of programming from those in the know: professional ST programmers. It's all here - in the world's first dedicated electronic ST programming magazine

RUNTIME 11

Create 3D graphics you'd swear were the real thing with our informative graphics series written by University lecturer Andrew Tyler. From first principles to extended programming practices, you'll be guided through graphic construction.

And to complement his series, RunTime features a variety of programs and code fragments which help to illustrate the concepts detailed in the magazine.

A couple of months ago we gave you HiSoft's DevPac so you could easily follow the examples provided by our guru of graphics. This month, we've included another batch of his program snippets so that you don't even have to type the stuff in! It's all here, on the CoverDisk, just waiting to be assembled and run.

GFA RESOURCE PROGRAMMING

Another instalment in our amazingly popular programming series detailing the vagaries of GFA resources and how to get the best from them.

Our expert, John Peters takes you by the hand on a guided tour of the requirements and prerequisites so that all those interested in the subject can follow and learn.

This month, John shows you how to create a useful control palette to manipulate colours when working in low resolution.

ABSTRACT DATA TYPES

Completing his two part article, Paul Overaa expands on his ADT queue theme initiated last month, reassessing the five available access routines and suggesting that to make use of such a construct it's a benefit for a programmer to use it without knowing anything of the internal arrangements.

Join Paul in his philosophical musings in RunTime.

STOS PROGRAMMING

Phil Lawson almost acquiesces to your numerous requests for the sprite designer and other goodies while attempting to continue his explorations into the STOS language. On offer: Solutions to the problems of meeting baddies, walking through acid pools, dying because of a lack of time and much, much more.

FIRST STEPS IN BASIC

It's back! Unfortunately, due to a slight technical hitch last month we were unable to bring you the latest gripping instalment of this introduction to BASIC. However, we've righted the wrongs and solved the unsolvable to reinstate the beginner's friend, Peter Phillips, to his rightful place.

CODE CLINIC

The hardy perennial with the evergreen outlook. Code Clinic is on hand every month to solve your coding problems. If your name is Robert Johnston, Nico Verwer or Chris Becker, whaddaya waiting for? Your problems have been solved...

GETTING STARTED

RunTime is a disk-based magazine program with articles providing programming help, hints for games writers and much, much more.

The magazine appears on side one of the Disk as RUNTIME.PRGM and all you need to do to read it is to double-click on it. You'll see a screen with the Atari ST User logo on the left and a list of menu options on the right.

These menus contain the titles of all the programs, articles, games and utilities on the CoverDisk. Click on a menu option and the article will be loaded and displayed on the screen. If your monitor isn't very clear you can print out the article and study it at your leisure.

At the foot of the screen you will find several icons. Here's what the main ones do when you click on them with the mouse...



This is the icon which enables you to return to the opening Main menu. Click this if you want to stop reading the current feature.



Use this icon if there is a particular page of the feature which interests you and which you wish to have on paper.



If you need to print the entire feature, tutorial and so on, this is the icon you need. Click here to send the complete feature to your printer.



Often, as you're reading a feature there'll be an associated picture to help illustrate a point. Click on this icon to see the picture.



Everyone, even learned programmers, needs help occasionally. If you find that some friendly advice is needed click the cross.

YOUR DISK NEEDS YOU

Here at Atari ST User, we're always on the look-out for new programming talent. ST owners the world over love to program their machines and we want to publish the fruits of those programming efforts - many of which rival commercial offerings.

If you've written a program which you think would be interesting to other readers, send it to us. Sure, it's easy to be discouraged, "they'll never publish this..." and you'd be right! We can't publish programs we don't receive, OK? Whatever language you've used, BASIC or assembler, Pascal or C, send it to us. We'll evaluate its potential and if it's good enough for the disk, we'll pay you handsomely to boot!

If you're going to send a program, here's what to do:

- Check that it works and that it's as bug-free as you can get it. May sound obvious, but you know your own code. An inexperienced user might press one wrong key and send the whole thing scampering away for cover.
- Make sure that it's ALL your own work. Don't rip off code from anyone else, then send it with your own name.
- Include full instructions.
- Include your name and address.
- If you want the disk back, include a stamped addressed envelope.
- Include a signed declaration bearing these words "I agree to indemnify Interactive Publishing against all possible litigation arising from the use of my software". If you're under 18, get your parent or guardian to sign it too. If you've already sent a disk in and haven't included a declaration, we'll send a form out to you before we use your program. And once you've sent a disk to us, be patient.

We get lots of submissions and we give each and every one careful consideration. Don't worry too much though if you have sent in something, with an SAE, and haven't heard anything since. Some things just take time and can't be hurried...

RUNTIME

**The definitive lexicon of programming from those
in the know: professional ST programmers**

Once again RunTime provides the answers to your programming problems.

CODE CLINIC

June's Code Clinic sees Mathew Lodge pondering the question of whether to use VDI or Line_A, solving some GDOS problems and helping with digitising through DMA.

FIRST STEPS IN BASIC

BASIC is always best for beginners and with this comprehensive tutorial, you'll soon be up there with the best of 'em, writing complex

programs in everyone's first programming language.

Peter Phillips continues his explorations of BASIC showing more of the all-important keywords and how to get the best from them in your own programs.

3D GRAPHICS

Busy boffin Andrew Tyler's been hard at it again to bring you the fourth in his great series about creating stunning three dimensional graphics on an ordinary ST.

From first principles the humble genius

has taken you to graphics of staggering complexity, and included on every CoverDisk there's a full selection of his much sought after programs – so that you don't even need to type them in!

STOS PROGRAMMING

That most popular of columns the STOS tutorial goes from strength to easily-programmed strength. This month there's further in-depth analysis of the maze game and more tricks, tumbles and turns to be had from the STOS language.

WRITING IT FOR RUNTIME

How many times have you read RunTime and thought that you could add some valuable programming knowledge to what's here already? We certainly have some expert writers offering their programming secrets in RunTime, but there's always something else to learn about and something else to discover with the ST. And that's why we need you.

That's right, if you have some in-depth knowledge of the ST – some little programming secret that you're sure others would benefit from, or some special way to achieve a particular effect – then we want to hear from you.

We're not chasing brilliant writers. If your copy is a little messy we can tidy it up for you. What we do require however, is a comprehensive understanding of the ST and the topic you're attempting to write about.

This is a fantastic opportunity for an unproven would-be ST writer to make his or her mark on the ST community. A chance in a lifetime. You'll have the honour of appearing in the world's first ST-exclusive programming magazine and we'll pay you handsomely to boot!

Interested? Then why not write to us here at RunTime? Send an outline of your feature ideas, together with a little background about yourself and copies of anything you've written (published or not) and we'll take it from there. Move it!

YOUR DISK NEEDS YOU

Here at Atari ST User we're always on the look-out for new programming talent. ST owners the world over love to program their machines and we love to publish the fruits of those programming efforts – many of which rival commercial offerings.

If you've written a program which you think would be interesting to other readers, send it to us. Sure, it's easy to be discouraged, "they'll never publish this..." and you'd be right! We can't publish programs we don't receive, OK?

Whatever language you've used, BASIC or assembler, Pascal or C, send it to us, we'll evaluate its potential and if it's good enough for the Disk – why, we'll even pay you!

If you're going to send a program, here's what to do.

- Check that it works and that it's as bug-free as you can get it. May sound obvious, but you know your own code. An inexperienced user might press one wrong key and send the whole thing scampering away for cover.

- Make sure that it's ALL your own work. Don't rip off code from anyone else then send it with your name. We'll send it back with the forces of law and order...

- Include full instructions.

- Include your name and address.

- If you want the disk back, include a stamped addressed envelope.

- Include a signed declaration bearing these words "I agree to indemnify Interactive Publishing against all possible litigation arising from the use of my software". If you're under 18, get your parent or guardian to sign it too.

This last point is very important and we cannot even consider your software without it. Then once you've sent a disk, be patient. We get lots of submissions and we give each one careful consideration.

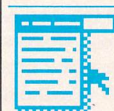
GETTING STARTED

RunTime is a disk-based magazine program with articles providing programming help, hints and much more.

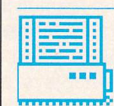
The magazine appears on Side One of the Disk as RUNTIME.PRGM and to read it just double-click on it. You'll see a screen with the Atari ST User logo on the left and a list of menu options on the right.

These menus contain the titles of all the programs, articles, games and utilities on the CoverDisk. Click on a menu option and the article will be loaded and displayed on the screen. If your monitor isn't very clear you can print out the article.

At the foot of the screen you will find several icons. Here's what the main ones do.



This is the icon which enables you to return to the opening Main menu. Click this if you want to stop reading the current feature.



Use this icon if there is a particular page of the feature which interests you and which you wish to have on paper.



If you need to print the entire feature, tutorial and so on, this is the icon you need. Click here to send the complete feature to your printer.



Often, as you're reading a feature there'll be an associated picture to help illustrate a point. Click on this icon to see the picture.



Everyone, even learned programmers, needs help occasionally. If you find that some friendly advice is needed click the cross.

RUNTIME

The definitive lexicon of programming from those in the know: professional ST programmers

From within the RunTime program, you can read the documentation for all of the programs on the Disk. Further documentation is also provided for some of the programs, either on the Disk or within these disk pages.

3D GRAPHICS

As Andrew Tyler's 3D graphics series continues, we include more assembler source code to help you follow the theory. No typing in long listings here, just run the example program, or load the sources into your assembler and enter the 3D world of computer graphics.

Even if you don't own an assembler there's no excuse - we've already given away version 1 of HiSoft's DevPac on the

CoverDisk! The files are in the EXTRAS.MAG folder on side one of the Disk.

MIDI MASTERCLASS

The Midi files for Ian Waugh's Masterclass are in the MIDI folder on side two of the Disk. Just load them into your favourite sequencer, attach your Midi equipment, and compose Kylie's next hit!

FIRST STEPS IN BASIC

Peter Philips continues looking at variable types. This month it's the turn of the string variable. More powerful than you may think, there are many uses for these, and Peter studies some of the more common, including MID\$, LEFT\$ and RIGHT\$.

CODE CLINIC

Code Clinic is the place where you get your questions answered, and where you can provide the answers to someone else's problem.

This month the topics covered include shuffling a deck of cards, STE enhancements, and running a program from within a desk accessory. Don't forget, if you have a problem, or if you can answer a problem in the clinic, write to Matthew Lodge.

STOS COLUMN

Due to a technical hitch (which means that the copy didn't arrive in time) we haven't a STOS column in this issue. But don't worry STOSSers, we won't be abandoning you.

WRITING IT FOR RUNTIME

How many times have you read RunTime and thought that you could add some valuable programming knowledge to what's here already? We certainly have some expert writers offering their programming secrets in RunTime, but there's always something else to learn about and something else to discover with the ST. And that's why we need you.

That's right, if you have some in-depth knowledge of the ST - some little programming secret that you're sure others would benefit from, or some special way to achieve a particular effect - then we want to hear from you. We're not chasing brilliant writers. If your copy is a little messy we can tidy it up for you. What we do require however, is a comprehensive understanding of the ST and the topic you're attempting to write about. This is a fantastic opportunity for an unproven would-be ST writer to make his or her mark on the ST community. A chance in a lifetime. You'll have the honour of appearing in the world's first ST-exclusive programming magazine and we'll pay you handsomely to boot!

Interested? Then why not write to us here at RunTime? Send an outline of your feature ideas, together with a little background about yourself and copies of anything you've written (published or not) and we'll take it from there. Move it send it to **Marc Young Europress Publications, Adlington Park Macclesfield, Cheshire SK10 4NP**

YOUR DISK NEEDS YOU

Here at Atari ST User we're always on the look-out for new programming talent. ST owners the world over love to program their machines and we love to publish the fruits of those programming efforts - many of which rival commercial offerings.

If you've written a program which you think would be interesting to other readers, send it to us. Sure, it's easy to be discouraged, "they'll never publish this..." and you'd be right! We can't publish programs we don't receive, OK? Whatever language you've used, Basic or assembler, Pascal or C, send it to us, we'll evaluate its potential and if it's good enough for the Disk - why, we'll even pay you!

If you're going to send a program, here's what to do.

- Check that it works and that it's as bug-free as you can get it. May sound obvious, but you know your own code. An inexperienced user might press one wrong key and send the whole thing scampering away for cover.

- Make sure that it's ALL your own work. Don't rip off code from anyone else then send it with your name. We'll send it back with the forces of law and order...

- Include full instructions.

- Include your name and address.

- If you want the disk back, include a stamped addressed envelope.

- Include a signed declaration bearing these words "I agree to indemnify Interactive Publishing against all possible litigation arising from the use of my software". If you're under 18, get your parent or guardian to sign it too.

This last point is very important and we cannot even consider your software without it. Then once you've sent a disk, be patient. We get lots of submissions and we give each one careful consideration.

GETTING STARTED

RunTime is a disk-based magazine program with articles providing programming help, hints and much more.

The magazine appears on side one of the Disk as RUNTIME.PRG and to read it just double click on it. You'll see a screen with the Atari ST User logo on the left and a list of menu options on the right.

These menus contain the titles of all the programs, articles, games and utilities on the CoverDisk. Click on a menu option and the article will be loaded and displayed on the screen. If your monitor isn't very clear you can print out the article.

At the foot of the screen you will find several icons. Here's what the main ones do.



This is the icon which enables you to return to the opening Main menu. Click this if you want to stop reading the current feature.



Use this icon if there is a particular page of the feature which interests you and which you wish to have on paper.



If you need to print the entire feature, tutorial and so on, this is the icon you need. Click here to send the complete feature to your printer.



Often, as you're reading a feature there'll be an associated picture to help illustrate a point. Click on this icon to see the picture.



Everyone, even learned programmers, needs help occasionally. If you find that some friendly advice is needed click the cross.

ARMALYTE

Configuration: All STs. Low resolution.
Colour only

Filenames: ARMALYTE - Folder
GAME.PRG - The program
TITLE.IFF - Data required by the program



RUNNING ARMALYTE

Armalyte can be run directly from the CoverDisk by simply opening the folder and double clicking on GAME.PRG. A joystick in the standard joystick port is required.

If you copy Armalyte to another disk, remember to include the file TITLE.IFF too.

In Armalyte you control the ship using the joystick. Holding the fire button down builds up the firepower of your weapon, releasing it lets fly! Don't forget to watch out for the extra pick-up weapons which can be collected along the way.

INSIDE THE STE

Configuration: All STEs
Filenames: STE - Folder

DEMO_A.S/DEMO_A.TOS - Source and executable example programs
BOMB.SPL,GUN1.SPL,GUN2.SPL - Samples used by demos
STE\STEPLAY\STEPLAY.PRG - Full STE Sound Sample Player program
STE\STEPLAY\STEPLAY.RSC - Resource file required by STEPLAY.PRG
STE\STEPLAY\STEPLAY.DOC - Program documentation

RUNNING INSIDE THE STE PROGRAMS

The programs can be run directly from the CoverDisk. The three demo programs are covered in this month's "Inside the STE" article. See the article and STEPLAY.DOC for details of using STEPLAY.PRG.

MIDI PROGRAMMING IN C

Configuration: All STs. Midi equipment required
Filenames: MIDIC - Folder

EAR.C/EAR.PRG - Source and executable example programs
JAZZ.C/JAZZ.PRG - Source and executable example programs
RANDOM.C/RANDOM.PRG - Source and executable example programs

RUNNING MIDI PROGRAMMING IN C

The programs can be run directly from the CoverDisk. Please note that the programs require Midi equipment attached to your ST using the Midi in and out ports.

The C source code is also provided for each program. This should be loaded into a suitable editor or wordprocessor for viewing and printing. See the article in the magazine for details.

TCB TRACKER MODULES

Configuration: All STs
Filenames: MODULE.MOD - Folder
TRANCE.MOD - A full TCB Tracker module

RUNNING TRANCE.MOD

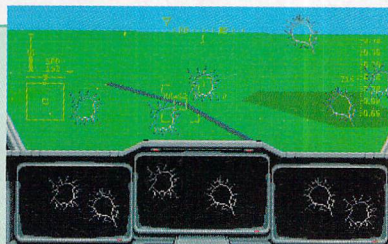
By popular request we have included another TCB Tracker module this month. This must be loaded into the TCB Tracker or the player program, both of which were included on our July CoverDisk.

BEYOND STOS

Configuration: All STs
Filenames: BEYONDST.OS - Folder
STOS01.BAS - Stos Basic source example program
FORMAT.ACB - Stos Basic disk formatter

RUNNING BEYOND STOS PROGRAMS

The programs must be loaded into Stos to be listed or run. Both programs are discussed in this month's Beyond Stos article.



RUNNING THUNDERHAWK

Thunderhawk *cannot* be run from the CoverDisk. You must copy THUNDER.TOS and THUNDER.DAT to the root directory (that is, not in a folder) of a blank, formatted disk. Once this is done, double click on THUNDER.TOS to run.

After the loading screen displays there is a small delay while the demo prepares itself. Click a mouse button as instructed on the intro screen, and now you're in control of Thunderhawk

Thunderhawk is a helicopter simulation sporting, to quote the advertising, "The fastest 3D graphic system to appear on any home computer". Well now you can judge for yourself. The demo contains a sample of some of the features of the full game, but your mission is simple: if it moves, shoot it. In fact if it stays suspiciously still for too long, shoot it anyway!

Nobody ever said that helicopters were easy to fly, and it may take you some practice to get the hang of this baby. Here are the controls:

Mouse left/right banks the 'copter left and right.
Mouse forward/backward banks you forward and backward.

Mouse left/right with right mouse button held

down rotates the 'copter while hovering.

Mouse forward/backward with right mouse button held down controls the engine speed, and thus your height/speed.

Left mouse button fires the currently selected weapon.

Clicking the left mouse button while holding the right mouse button down changes the currently selected target.

Right mouse button changes the selected weapon. To do this the right button must be clicked quickly, and not held down too long.

Your available weapons as shown in the Head Up Display are:

- GUN - Aim and Fire
- FFAR - Aerial rocket Aim and Fire
- MWAR - Multi-warhead Aerial rocket Aim and Fire
- MK-82 - Unguided Bomb
- MK-83 - Unguided Bomb
- AGM-214 - Guided Air-to-Ground missile
- AGM-65P - Guided Air-to-Ground missile
- AIM-10B - Air-to-Air heat seeking missile
- AIM-11F - Air-to-Air radar guided missile
- F1 Operate the radar jammer
- F2 Operate the infra-red jammer

THUNDERHAWK

Configuration: All STs. Low resolution. Colour only
Filenames: SIDE2\THUNDER - Folder

THUNDER.TOS - The program
THUNDER.DAT - Data required by the program



C Turn on long range target camera

F Show view from the currently selected target

G Show view from currently selected weapon

H Show view from cockpit

M Show overhead "satellite" view. use + and - on keyboard top row to zoom in and out.

P Pause

Use the numeric keypad for an external view of your chopper!

Enter selects external view.

2,8,4,6 rotate the view over and around

1 and 3 rotate in 90 degree steps

Zero positions the camera directly behind you

5 positions the camera directly above you

7 and 9 zoom in and out

To aid you in your mission, this demo does not allow you to crash. However you can still be shot down! To take off safely, hold down the right mouse button and push the mouse gently forward until you hear the rotor speed increase.

Watch the horizon and the height indicator to the right of your HUD. As soon as you take off, release the right mouse button. Move the mouse to stop any forward or sideways movement. You may need to increase the throttle a little to gain some more height, then you're away.

PICTURE GALLERY

Configuration: All STs. Low resolution. Colour only

Filenames: SIDE2\GALLERY- Folder

SHOWTRIX.PRГ - The gallery program

*.PC1 - A number of pictures for the gallery program to display

RUNNING PICTURE GALLERY

The Picture Gallery program may be run directly from the CoverDisk by double clicking on the program SHOWTRIX.PRГ.

It will load and display all pictures in a number of different formats from the folder from which the program was started. Use the function keys to increase or decrease the length of time each picture is displayed, and use the Q key to quit.

Showtrix is a neat picture gallery program which uses various different methods to fade between consecutive pictures. A number of different file formats are catered for, so you can almost certainly use it with your own pictures produced with your favourite art package.

CONVERT

Configuration: All STs. All resolutions.

Filenames: CONVERT - Folder

CONVERT.DOC - Program documentation

CONVERT.ACC - The program

CONVERT.RSC - Program resource file

RUNNING CONVERT

Convert is a desk accessory, and will only run if it and its resource file are copied to the root directory (that is, not within a folder) of your boot disk. Once this is done, boot your ST. Convert is now accessible by using the Desk menu from the Desktop.

Convert is a programmers' tool which will convert between Decimal, Hex and Binary. Being a desk accessory it allows the programmer to use it from within any Gem application, such as a program editor.

Simply select Convert from the Desk Accessory menu, select the input value type from Hex, Dec or Bin, type in your number, then select the output type. Your number will be immediately converted.

DR FOX'S 1990 FOOTBALL ANNUAL

Configuration: All STs. Medium resolution. Colour only
Filenames: SIDE2\FOOTBALL.FOX - Folder

FOX_DEMO.PRГ - The demo program

READ_ME - Instructions for using the program

ENGAMES.DAT - Data required by the program

SEASON90.DAT and

SEASON91.DAT - folders containing data for last and this season.

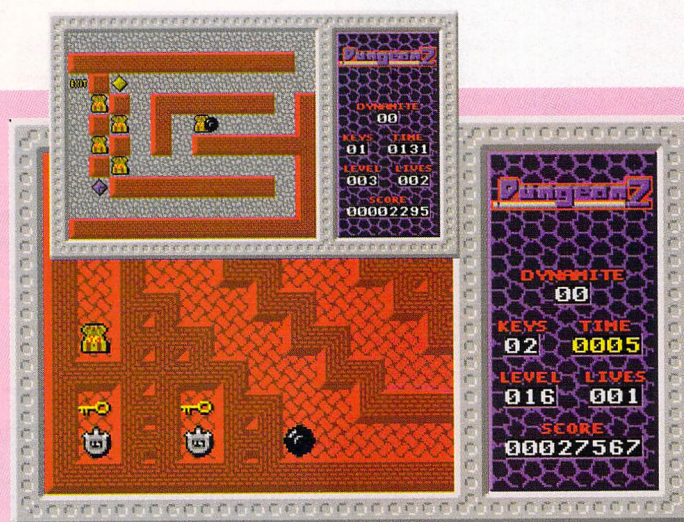
RUNNING DR FOX'S FOOTBALL ANNUAL

Although the program can be run directly from the CoverDisk it is recommended that all the files within the FOOTBALL.FOX program are copied on to a blank formatted disk before running, as data may need to be saved.

Read the program documentation by double clicking on the file READ_ME and selecting the Show option from the Alert box which appears.

Dr Fox's Football Annual allows you to keep details of the results of football matches throughout the season, and to analyse them in a number of different ways.

Full details can be found in the READ_ME file.



Now how do I get those keys when I ain't got no Dynamite?

DUNGEONZ

Configuration: All STs. Colour only

Filenames: SIDE2\STOSGAME\DUNGEONS - Folder

DUNGEONZ.DOC - Program documentation

DUNGEONZ.PRГ - The program

LEV.DAT - Folder containing data for the program

RUNNING DUNGEONZ

Dungeonz can be run directly from the CoverDisk by simply opening the folder and double clicking on DUNGEONZ.PRГ. A joystick in the standard joystick port is required.

To read the documentation on disk, make sure you are in medium or high resolution, then double click on DUNGEONZ.DOC and select Show from the Alert box that appears.

If you copy DUNGEONZ to another disk, remember to include the folder LEV.DAT too.

Dungeonz was written in Stos Basic by Ben Clark. The basic aim of the program is to guide Bertie the Bouncing Bomb around the Dungeons of Zorkrath (which is just outside Leicester, I think), collecting all the gems before reaching the exit.

There is a time limit, and the time remaining is shown on the right of the screen, together with the number of lives, the amount of dynamite Bertie is carrying, the number of keys in his possession, the current level number and that all-important score.

When you have collected all the gems, Bertie will start to glow. At this point reaching an exit will transport you on to the next level. Be warned, though, there are some tricky moments in this game and, as Ben says, "levels aren't always what they seem".

MAZE

Configuration: All STs. Low resolution. Colour only. Two joysticks required

Filenames: MAZE - Folder

MAZE.PRГ - The program

MAZE01.NEO, MAZE02.NEO,

MUSIC.DAT - Program data file

CHASE.MZE, ORIGINAL.MZE, THE-

MAZE.MZE, TINY.MZE - Example

mazes

README.DOC - Program documentation

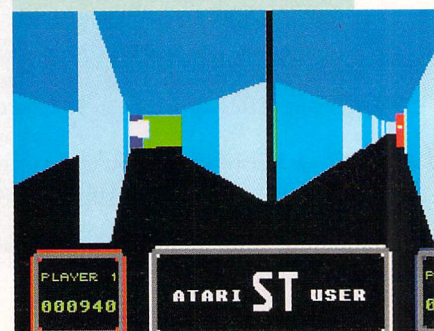
RUNNING MAZE

To read the program documentation, just double click on README.DOC, then select Show from the Alert box that appears. To run the program, simply double click on MAZE.PRГ.

Maze, written by John Parker and Martin Denmet, is a foreboding three dimensional, two player, one-helluva-maze-game. With a split screen two players can rush round the maze, either just to find the other player, or to shoot them!

Supplied on the CoverDisk with four full mazes, it includes its own maze designer, which allows you to load and save any number of mazes, and test them on a full screen.

Read the program documentation for details on using the program and the maze editor.



Mike Dixon-Kennedy looks at disk formatting from Stos

- #09 - Write a sector
- #10 - Format a track
- #18 - Produce a boot sector

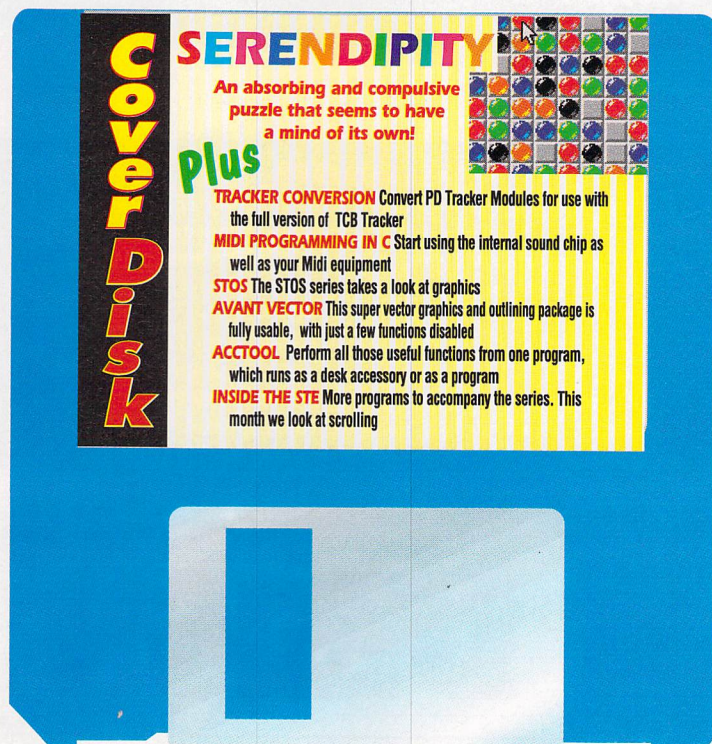
- 1 Lay down the tracks and sectors on the disk.
- 2 Produce a boot sector in ram.
- 3 Write boot track to disk.

\$87654321 The constant \$87654321 must be used or formatting will not

Table 1

COVERDISK

This month the CoverDisk is absolutely crammed full of the very best software: great games, the ultimate utilities and truly devastating demos. Read on!



SIDE ONE

Obitus

If you go down to the woods today, you're in for a big surprise. And we're not talking Teddy bears here!

Serendipity

Serendipity (se-ren-dip-ity) n. the faculty of making happy discoveries by accident. You'll become less than serene once the program starts insulting you!

STOS

All the programs to support our STOS series to save your weary typing finger(s).

Module Converter

If you've created song modules using the PD version of the TCB Tracker, then this utility is the one for you. It converts between the PD format and the commercial format, as given away on our July CoverDisk.

Midi Programming in C

The programs supporting this series can be found in full on the disk.

Avant Vector

A working demo of this superb vector drawing package, with only the Save option disabled.

SIDE TWO

AccTool

A multifunction utility which lets you perform almost any function you care to name, and it runs as a desk accessory or a program.

Inside the STE

If you've got an STE, then this is where to look for the programs accompanying the series.

Obitus

Configuration: All STs. Low resolution.
Colour only.

Filenames:
Folder: OBITUS
OBITUS.TOS - The Demo Program

Running Obitus

Running Obitus couldn't be much more simple. Just double click on OBITUS.TOS (You

did make sure you were in low resolution, didn't you?) then use the mouse to select the required action from the menu at the bottom of the screen.

Obitus finds you "lost and alone in a world not your own" where "Trolls, dwarves, knights wizards and kings infest the four shires." It all sounds remarkably like the ST



The sanctuary of the Tower lies beyond the door, while the danger of the forest waits

Module File Converter

Configuration: All STs. Mono or colour
Filenames:

Folder: CONVERT.MOD
CONVERT.PRg - The Module Conversion program
CONVERT.TXT - Instructions for using the program
CONVERT.RSC - Resource file for the program.

Running the TCB Tracker Module Converter

To run the program, simply double-click on CONVERT.PRg. Follow the onscreen prompts to convert your modules.

The public domain demo of the TCB tracker used a slightly different file format to the commercial version, as given away on our CoverDisk in July. Frustrated Trackers need fret no longer, as this program will convert from the PD file format to the new file format, for both MOD and SNG files.

Midi Programming in C

Configuration: All STs. Midi equipment required.

Filenames: Folder: MIDIC

CHORDORG.C, INFERNAL.C, STYLOPHO.C - C Source code
CHORDORG.PRG, INFERNAL.PRG, STYLOPHO.PRG - The compiled programs
README.DOC - The article accompanying this month's programs

Running Midi Programming in C

The programs supplied here accompany the Midi Programming in C series. This month's final article is on the disk (README.DOC). If you are going to make any changes to the programs, they should be copied to another disk first.

Beyond STOS

Configuration: All STs. Colour only

Filenames Folder: STOS PROGRAM.BAS, SUBROUT.BAS - The STOS Programs
PIC.NEO - Example screen required by the programs

Running Beyond STOS

The programs supplied here accompany the Beyond STOS series in the magazine, and can only be used by loading into STOS. If you are going to make any changes to the programs, they should be copied to another disk first.

Full instructions can be found in the Beyond STOS pages of the magazine.

Inside the STE

Configuration: STEs only, colour only

Filenames: Folder: SIDE2\STE

SCROLL.S, WOBBLE.S - The Assembler source code to accompany the series
SCROLL.PI1, WOBBLE.PI1 - Example screens required by the programs
SCROLL.TOS, WOBBLE.TOS - The assembled version of the programs
STEPAL - Folder containing STE Palette editor desk accessories

Running Inside the STE

The programs supplied here accompany the Inside the STE series in the magazine and can only be run on STE computers. If you are going to make any changes to the programs, they should be copied to another disk first.

Full instructions can be found in the Beyond STOS pages of the magazine.

The Files in the folder STEPAL should be copied to the Root Directory (ie not in a folder) of your boot disk in order to be available. See the file STEPAL.DOC for full information. This can be read by double clicking on it, then selecting Show from the Alert box which appears.

Serendipity

Configuration: All STs. Low Resolution. Colour only.

Filenames: Folder: SRNDPTY

SRNDPTY.PRG - Program itself

SRNDPTY.SCR, SRNDPTY.SND - Data files required by the program

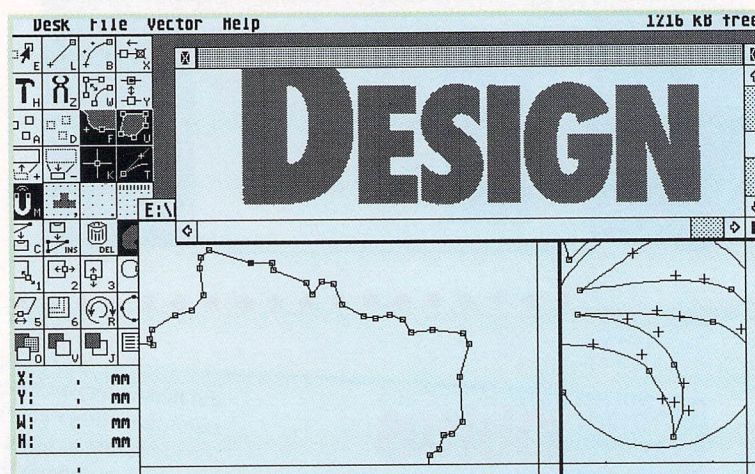
READ.ME - Program documentation

SOURCE - Folder containing the C source code for the complete game

Running Serendipity

Make sure your ST is in low resolution, then double click on SRNDPTY.PRG. Help is available by clicking on the Info and Help buttons.

Full instructions can be found within the program, but do make sure you try every option to get the best response from the program! Oh, and you wouldn't dream of pressing the button labelled "Don't click", would you?



Avant Vector

Configuration: All STs. Mono only

Filenames:

Folder: SIDE2\AVANT

AV_HELP.TXT - Information and tutorial for using this demo.

AV_DEMO.PRG - The demo program itself

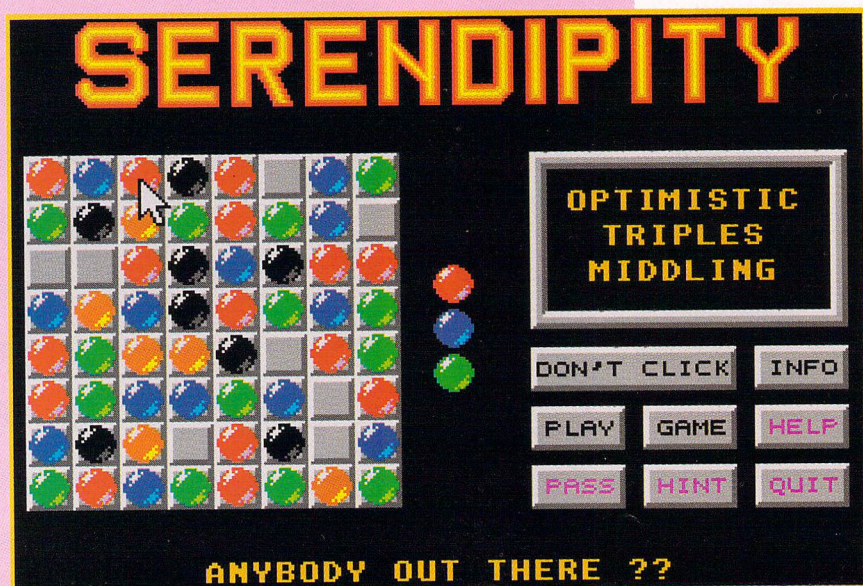
DESIGN.IMG, FRANCE.CVG

OASIS.CVG - Example graphic files to use in Avant Vector.

Running Avant Vector

Full documentation on running Avant Vector is the file AV_HELP.TXT. Double click on this file, then select Show from the Alert that appears to read the instructions. For help during the program, right-click on an icon, or point and press the Help key.

Avant Vector allows you not only to manipulate vector graphics, but to create vector graphics automatically from a bit-image graphic. The demo supplied here is fully functional except for the Save and Print options.



With Serendipity the computer not only plays a mean game, it also insults you if you take too long or make a mistake

Mike Dixon-Kennedy finishes with Traps, and begins to delve into the colourful world of graphics

I'll leave you this month working out just how you can use the `PACK` command to the best advantage. Next month we'll take a look at some of the other graphics commands that aren't too well explained in the manual by making a start on an art package that will come as a series of

In GAMER this month

Knights of the Sky



Another World



Heimdall



ATARI ST USER

EDITOR John Butters
ART EDITORS Terry Thiele
Tym Leckey
TECHNICAL EDITOR Andrew Wright
PRODUCTION EDITOR Phil Morse
GAMER EDITOR Daniel Whitehead
STAFF WRITERS Jonathan Maddock
Darren Evans
GROUP AD MANAGER Jane Conway
SENIOR SALES Lisa Bracewell
ADVERTISING SALES Ruth Priestley

CUSTOMER SERVICE Sandra Childs
Linda Thomas
Sirje Green
Brigid Ruddick
Robert Matthews

PRODUCTION MANAGER Sue Cantrill
SYSTEMS MANAGER David Stewart

CIRCULATION MANAGER David Wren
DISTRIBUTION COMAG (0895) 444055
SUBSCRIPTIONS 051-357 2961

Member of the Audit Bureau of Circulations

ABC 40,450
July-Dec 1991

Published by Euromag Publications Ltd.
Europa House, Adlington Park,
Macclesfield SK10 4NP

Tel: 0625 878888 (All departments) Fax: 0625 879966

PUBLISHER Richard Williams
ASSISTANT PUBLISHER Eddie McKendrick

CHAIRMAN Derek Meakin
MANAGING DIRECTOR Peter Glover
FINANCE CONTROLLER Ian Bloomfield

We regret Atari ST User cannot offer technical help on a personal basis either by telephone or in writing. All reader enquiries should be submitted to the address above for possible publication.

Atari ST User is an independent publication and Atari are not responsible for any of the articles in this issue or for any of the opinions expressed

© 1992 Euromag Publications Ltd. No material may be reproduced in whole or in part without written permission. While every care is taken, the publishers cannot be held legally responsible for any errors in articles, listings or advertisements.

euromag
PUBLICATIONS

For over six years Atari ST User has been the leading magazine for ST enthusiasts. As a key member of the Euromag magazine group, Atari ST User promises to inform, educate and entertain its readers each month with the most dedicated coverage of the ST available.

Atari ST User • Amiga Computing • PC Today
Shareware Shopper • The Micro User
Games-X • Amiga Action • ST Action
Computer Trade Weekly • Crash
Zzap!64 • Sega Force

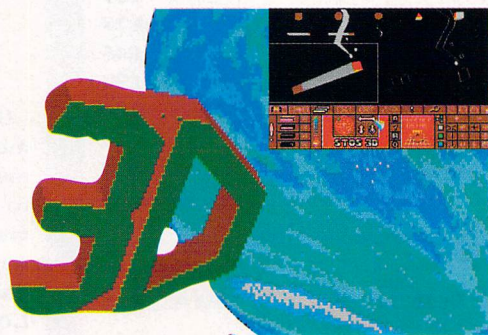
REVIEWS

Calamus SL

With a new version and a new distributor, is this DTP package still top of the pile?48

Stereo Master

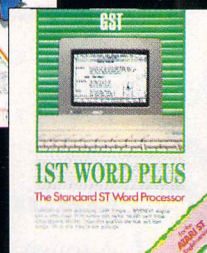
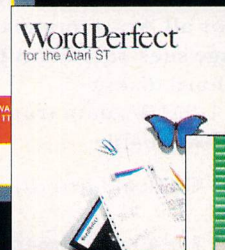
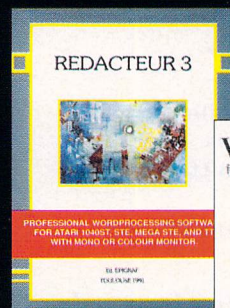
A versatile and usable sound sampling package that won't break the bank?53



STOS 3D

Harness the power of full 3D vectors with this amazing programming language62

COVER FEATURE



Wordprocessing
The ultimate guide to text crunchers – what they do, what's available and what the jargon means57

THE COVERDISK

We've got a real show-stopper for you this month!

- A world exclusive – *Mutant Caterpillars* is an exciting and revolutionary 3D game written using the remarkable new STOS 3D programming language
- Master the basics of wordprocessing with *First Word* and our step-by-step guide in this month's in-depth guide to wordprocessing
- Midi programmers will appreciate two useful utilities, *Panic Button* and *Local Control*
- Get drooling over *Turbo 4*, one of the fastest reset-proof RAM disks around!



The DISK starts on page 27

aspects

midi and music

No serious ST musician can afford to ignore the best music column in the business!

108

emulation

The column that exploits the ST's chameleon-like tendencies to the full!

111

communications

The world of modems and bulletin boards – mass communication for the 90s.

113

desktop publishing

Producing top-quality newsletters and posters is easy with your ST and this column.

115

CoverDisk

Two great demos head the cast of this month's CoverDisk show. Curtain up...

END OF AN ERA

Single sided users will immediately notice that they cannot access the CoverDisk at all this month. Sorry folks! The rather strange format we used in the past considerably restricted the size of the programs we were able to put on the CoverDisk.

However, each individual reader is important to us. We'll copy the files on to two single sided disks and return them to you – along with some advice on how to get your internal drive upgraded...

If you have a single sided drive, send your original CoverDisk with a cheque for £1.50 payable to Atari ST User, to:

SSD Service
Atari ST User
Europa House
Adlington Park
MACCLESFIELD
SK10 4NP

Problems with your CoverDisk?

Hotline Number: 0780 55604

With thousands of disks being duplicated each month, a few will inevitably be faulty. Fortunately we can help – send the faulty disk to:

Stanley Precision Data Systems,
Unit F, Cavendish Courtyard,
Sallow Road, Weldon North
Industrial Estate, CORBY NN17 1JX

A replacement will be sent free of charge.

If you have any other problems with the CoverDisk, our technical editor, Andrew Wright, will be available on the Atari ST User CoverDisk hotline between 3.30pm and 7.30pm every Monday. Please do not call outside these hours as he will be unable to answer your query.

Using the CoverDisk

In order to cram as much as possible on to the CoverDisk, some of the files have had to be archived. To help identify these programs, they have all been named with an initial letter X. For example, X_ADDRSS.TOS is the archived ADDRESS database.

To extract and run programs like this, you will need to format a blank disk, copy the X_?????.TOS file across to this disk and then double click on it. The files will be displayed onscreen as they are extracted. Once the process is finished you will see the unarchived program and associated files on the disk.

MUTANT CATERPILLARS - THE STOS 3D DEMO

A stunning demo of the remarkable new STOS 3D Basic programming language and a complete action-packed game all rolled into one!

Ever fancied being a 3D programmer? We challenged Mandarin Software to come up with a playable demo game created using the new version of STOS 3D which would demonstrate the ease of use and flexibility that has turned STOS Basic into the most popular programming language on the ST.

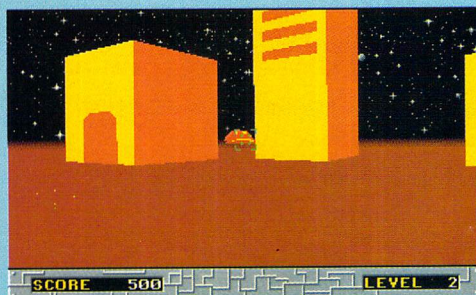
To really turn the screw, we gave them just two days to do it in! Mutant Caterpillars is the result, showing what a competent programmer can do in such a short space of time.

The game can be run straight from the CoverDisk. Plug in a good joystick and double click on CATA.PRG for some amazing 3D action!

The object of the game is to prevent the mutant caterpillars – the funny looking red things with cheerful faces – from reaching the 3D village just in front of you. You can navigate around the village using the joystick controls and blast any caterpillar within range using the Fire button.

The background can be toggled off to suit your preference using the F1 function key, as can the radar (the series of small dots in the lower left-hand corner of the screen) using F2.

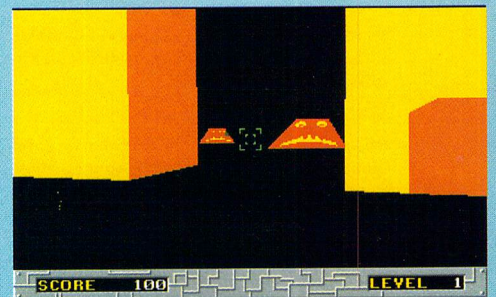
Enjoy!



It might only be three pokey little buildings, but it's home to someone



Get the mutant caterpillars before they take over the whole of your 3D village!



It's here at last – the long-awaited 3D extension to the STOS Basic programming language

Configuration: All STs (except Mega STs), low-res colour
Filenames:

STOS.3D – Folder
OBJECTS – Folder containing 3D objects
CATA.PRG – Game program
C3D.PRG – STOS 3D program

Until now, if you wanted to program 3D graphics you had to write in assembler to get the speed required. You also needed to do an awful lot of work to get even the most simple 3D shape onscreen. Wouldn't it be nice, though, if you could display 3D shapes on the screen using simple commands in Basic in the same way that sprites can be displayed?

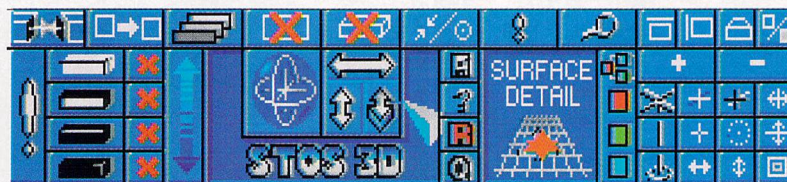
With the release of STOS 3D, this is exactly what you can do! Created by Voodoo Software, STOS 3D is an extension to STOS which gives an additional 30 commands dedicated to the display and manipulation of 3D objects.

As it comes as an upgrade to STOS, once installed you can use the 3D commands in any program you like, and even compile them for extra speed.

Comprehensive

STOS 3D comes on two disks, together with a comprehensive manual covering the new commands and utilities. Installation is a relatively simple matter of booting from the installation disk and following the prompts.

As STOS 3D can only be added to STOS v2.6 or higher, an upgrade is supplied to update earlier versions. If present, the compiler can also be updated to allow compilation of the



The control panel sits at the bottom of the screen and contains all the icons you need to design your 3D objects

STOS 3D commands. In the same way that you need a sprite editor to design the sprites to be used in STOS, you need a 3D Object Modeller to design the 3D objects to be used in STOS 3D.

This is provided as a standalone program, and comes complete with a number of examples for you to play with to see just what can be achieved with the software.

The manual is an A5-sized booklet which manages to cover all of the new commands, and the Object Modeller, in just over 100 pages.

The first couple of chapters introduce the concept of 3D programming in general. The next few chapters take you through backing up your disks, and upgrading your current version of STOS, if necessary.

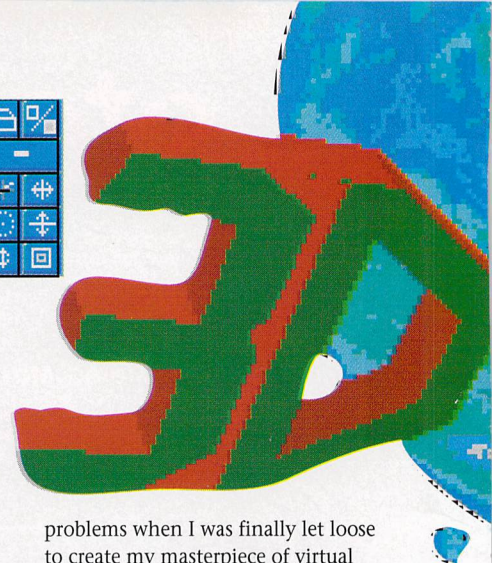
Now, at last, you are ready to jump into the world of 3D. The manual

describes the Object Modeller by example, taking you through the steps required to manipulate 3D objects. The next chapter goes into the Object Modeller in more detail, describing the more complicated procedures available.

Diagrams

Finally, by chapter eight – more than halfway through the manual – we're finally ready to start doing some programming. This chapter starts by describing the concept of 3D space, using diagrams well to explain the x, y and z co-ordinate system used by STOS 3D.

It then introduces the new commands using example programs supplied on the disk to help with the tutorial. Overall, I was impressed by the manual, and found that by working my way through it, I had very few



problems when I was finally let loose to create my masterpiece of virtual reality!

A major headache encountered by all programmers of 3D games is the design of the objects to be used. Whereas with sprites the sprite editor is generally a fairly simple program, with three dimensions to play with, the Object Modeller can be somewhat complex.

With the STOS 3D Object Modeller program, however, a lot of thought has obviously gone into the user friendliness, making it almost intuitive.

The workscreen of the Object Modeller is very simple. The lower third is filled with the control icons. These give access to all of the functions while the objects you are working on occupy the top two-thirds of

Marc Young steps into the world of 3D programming the easy way with STOS 3D

3D made easy

Programming in STOS 3D

Once you manage to tear yourself away from the Object Modeller you'll want to use the objects you have created in your own programs. And it honestly couldn't be simpler. Take the following program:

```
10 td init 50000
20 td load "rocket"
30 td object 1,"rocket",0,0,1000,0,0,0
40 td redraw logic
```

Line 10 simply reserves 50k for the 3D system. This actually isn't necessary unless you are going to compile your program.

Line 20 loads the 3D object called ROCKET from the currently selected 3D object folder. Although this may actually involve loading more than one file, this is transparent to the programmer as STOS 3D handles everything for you.

Line 30 defines object number 1 as being a rocket – that is, using the 3D object we just loaded – and places it at the centre of the screen with no

rotation. Finally, line 40 actually draws the screen. The first thing to notice is that all the commands begin with "td". This does make life easy when you're debugging a large program – if nothing else you know which manual to use to look up the command syntax!

Next, notice that line 30 describes object 1 as being the rocket. All objects – there can be a maximum of 20 in play at one time – must be assigned a number, but any number of copies of it can be used without having to load multiple copies of the same object, which saves a huge amount of memory.

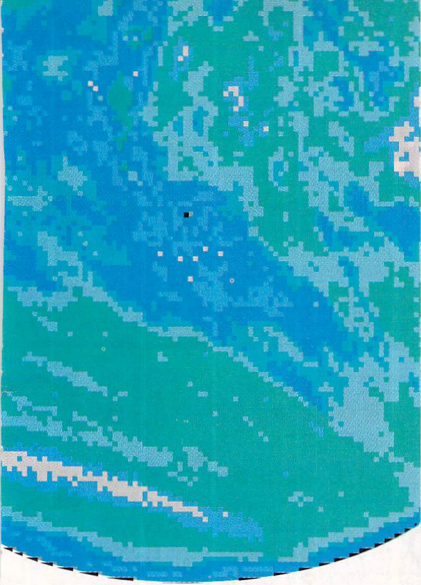
It is also worth noting that line 30 does not actually draw the object, it merely defines its current position. So many objects can be loaded and defined before actually drawing any of them.

The figures used for defining the position of an object in our 3D world are the x,y and z co-ordinates, and the A,B and C angles. Most of you will be familiar with x,y and z co-ordinates, and the angles

merely define the object's rotation about each of these axes. The origin of our world lies at 0,0,0, so decreasing the x co-ordinate to -100, for example, would move the object left by 100 units. Adding 100 to the z co-ordinate moves the object "into" the screen by 100 units.

Although there is enough here to define a 3D world and move things about within it, this is only the beginning! A special object, object 0, is defined as being your viewpoint. This means that while things are going on in your 3D world, you can move your point of view about like a camera, zooming and panning to your heart's content.

In addition to the basic commands allowing us to move an object (td move) or change its angle (td angle) there are commands which allow relative movement – defining speed – return its position, point objects at one another, test whether an object is visible, test for collisions between objects and even animate both the 3D objects themselves and the patterns superimposed on their surfaces!



the screen.

The top two-thirds are split into 12 "shelves". The first ten shelves are arranged in two rows of five, the lower row of which is initially empty and, therefore, invisible. The top row contains five standard shapes, including a cube and a pyramid.

The second row of five can be used to store your own range of "standard" objects. In this way, large 3D designs can be quickly built using a small number of basic building blocks.

The last two shelves are used for editing 3D objects. Here we can design building blocks or final objects and manipulate them in a number of ways.

Creation

Describing how a simple object is created is probably more useful than giving a step-by-step account of STOS 3D's features. In fact the manual takes you through creating this in some detail, so you will see how simple it really is to create your own 3D designs.

For the example, a simple T shape is created. Obviously this comprises two long, thin rectangular blocks.

Step 1 Copy the cube from the top shelf into the work shelf. This is achieved by clicking on the top shelf required, clicking on the work shelf required, then clicking on the Copy icon.

Step 2 The block needs to be

stretched vertically into a column. This is simply a matter of making our cube a "group" by clicking on the group icon, then selecting the Vertical Stretch icon and dragging until the shape looks OK.

Step 3 Copy our stretched cube into the other work shelf using the Copy tool. Now select the top face of the first column – using the Face Select tool – and the side face of the second column, using the same tool. Both faces are now highlighted.

Step 4 Select the Glue icon. This takes the two columns and glues the two selected faces together.

Step 5 Tidy the object up by selecting the Priority icon, which ensures that the objects are drawn in the correct sequence. You don't have to tell the program what the correct sequence is – it works it out.

And that's it! you've just created a 3D object! It really can be that simple. As you come to use more of the features available, and get used to them, you will find that it takes very little effort to create some quite complex looking objects. One added extra which really can make a big difference to the final outcome is the Surface Detail option. A small grid is displayed, over which a shape can be drawn. This shape can then be applied to any face of any object. So, for example, you could put a company logo on a racing car, or the national flag onto an plane or spaceship.

If you own STOS, then STOS 3D could be the best thing that's happened to your ST in a long while. If you don't own STOS, then STOS 3D is a damned good excuse to rush out and buy a copy!

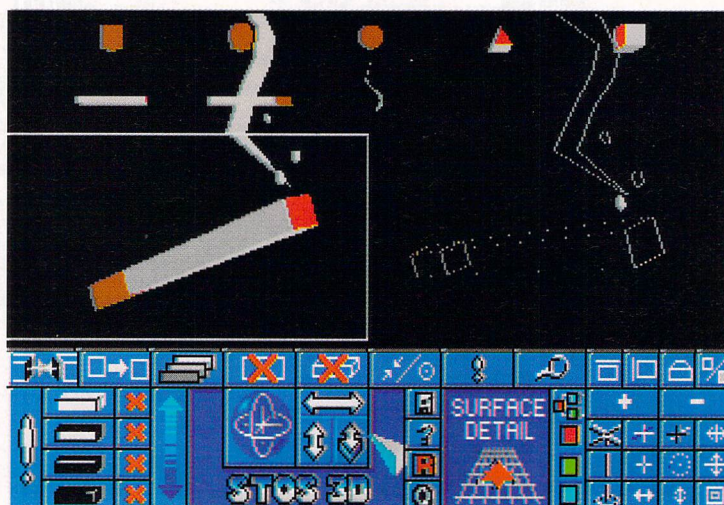
Product: **STOS 3D**
 Supplier: **Europress Software,**
Europa House, Adlington Park,
MACCLESFIELD SK10 4NP
 Telephone: **0625 859444**
 Price: **£34.95**
 Configuration: **All STs (except Megs),**
colour monitor

Once you've created a few objects – or borrowed some of the examples supplied – it only takes a few minutes to have them shooting around the screen



The new commands

- td ADVANCED** – allows access directly to the object structures in memory
- td ANIM** – animate an object by moving points on it to an absolute position
- td ANIM REL** – animate an object by moving points on it relative to their previous position
- td ANGLE** – set an object's orientation by specifying angles A,B and C
- td ANGLE A** – obeys a string of commands to change the objects angle, in a similar way to the STOS sprites MOVE command
- td ANGLE REL** – set an object's orientation relative to its current orientation
- td BACKGROUND** – display a picture in the background
- td BEARING** – returns the bearing of any object from any other
- td CLEAR ALL** – remove all loaded objects from memory
- td CLS** – clear the current 3D screen
- td COLLIDE** – determines whether objects have collided
- td DELETE ZONE** – remove a zone
- td DIR** – specify the Directory containing 3D objects
- td FACE** – rotates an object to face another, and returns its bearing
- td FORWARD** – move the object in the direction it is pointing
- td INIT** – reserve a number of bytes for 3D
- td KILL** – remove an object
- td LOAD** – load an object
- td MOVE** – move an object to the specified x,y,z co-ordinates
- td MOVE REL** – moves an object relative to its current position
- td MOVE X** – obeys a movement string in a similar manner to STOS sprites. **td MOVE Y** and **td MOVE Z** are also available
- td OBJECT** – defines an object's initial position and attitude
- td POSITION** – returns an object's current position
- td PRIORITY** – define the order in which objects will be drawn
- td RANGE** – returns the range of an object from another
- td REDRAW** – draw all 3D objects defined
- td SCREEN** – converts a 3D co-ordinate into a screen x,y co-ordinate
- td SCREEN HEIGHT** – allows size of screen required to be defined
- td SET COLOUR** – define an object's colour combinations
- td SET ZONE** – define a sphere around an object for collision detection
- td SURFACE** – copy a surface pattern for animation
- td SURFACE POINTS** – change the way a surface is copied
- td VIEW** – converts World co-ordinates to Local co-ordinates
- td VISIBLE** – returns whether an object is visible or not
- td WORLD** – converts Local co-ordinates to World co-ordinates
- td ZONE** – return information about a zone



What a drag! I'm puffed out after designing this 3D extravaganza. This screenshot shows the development of the flag, with individual elements being designed and held in the top shelves before being combined to form the final object

Bottom line

FEATURES: Everything you wanted to know about 3D graphics, even if you were afraid to ask

EASE OF USE: Easy for even the novice to produce sparkling graphics programs

Features	Ease of use
Excellent	Excellent
Good	Good
Average	Average
Bad	Bad
Appalling	Appalling

specials

16 STOS 3D tutorial

We guide you through the installation and basics of STOS 3D, given away free on this month's CoverDisk

28 Free for all

The definitive guide to the public domain and shareware – what it is, what's best and whether you should register your software

40 It's all in hand

All you need to know about the mice, trackballs and joysticks that can control your ST

64 Word of advice

Get more from your wordprocessor with this comprehensive guide to the features that are often missed

reviews

58 Prism Paint

This new paint and animation package is found to be one of the best designed programs in its class

60 Easy Text Professional

This excellent value-for-money document processor has just been improved and we say if it should be added to your collection

63 Warp 9

A software-based screen accelerator with plenty of useful features is put through its paces

69 Chronos 3D

A brilliant piece of software that should make life much easier for animators



91 There's £1,300 – worth of World War II books, videos and software up for grabs in this spectacular Campaign competition

regulars

9 News

Comdex sees new products for Falcon, but is the computer available?

12 CoverDisk

Find out how to use the amazing software we've given away on this month's disk

37 Write Now

The pages where you comment, debate, criticise and praise the topics of your choice

44 Gallery

We show some of the best artwork that has been flooding in to Gallery from our readers during the last few months

49 STOS 3D

A new tutorial which will show you how to get the very best from this versatile programming language

53 Advice Service

Hard drive problems, boot up failures and all types of strange goings on. We solve these problems and more

56 Public Sector

Seven new utilities and games from the public domain are tried and tested. Discover the best

100 Subscriptions

Take advantage of our great offers – and guarantee your copy

102 Aspects

Six pages of applications for your ST

106 Classifieds

Buying or selling, it's the only place to advertise

The Competitions

108 Fancy a £300 spree at a computer centre? Maybe ST User and First Computer Centre can help...

The CoverDisk



Containing the full working version of **STOS 3D**, the essential extension to the **STOS** programming language that will enable you to write spectacular 3D games

PLUS:

● **GMAP** - a disk mapping utility to enable you to check and analyse data on your disks for fragmentation

● Linked with this month's main feature, **Ditch Day Drifter** is an exciting adventure game based on life in a college campus

● **YMG125** is a new and fast communications transfer protocol that claims to be faster than the rest for downloading and uploading files

● Never miss an important appointment again. The **Alarm** is a useful desk accessory that will help you keep track of the time

The DISK starts on page 12



COVERDISK

This month's disk contains an incredible giveaway – the

STOS

Create your own 3D worlds and games with this extension to the very popular STOS programming language

Programmed by: Anthony Wilkes and Richard Lewis
Configuration: All STs (1Mb recommended) – lo-res
Filename: X_DEMO1.TOS, X_DEMO2.TOS, X_INSTL1.TOS, X_INSTL2.TOS – All self-extracting archive files

Although you can use STOS 3D on its own to create and view 3D objects, you will require STOS Basic in order to develop games which use such objects.

Also, although the Object Modeller will run in S12k, owners of STs with less than 1Mb are advised to upgrade to 1Mb to take full advantage of STOS 3D's powerful features.

STOS 3D games

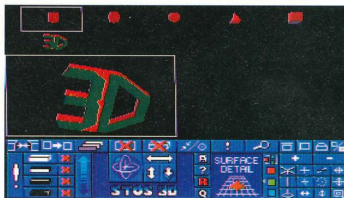
Once you have mastered STOS 3D programming and created your first 3D game, why not send it in to us for evaluation and possible inclusion on the CoverDisk? If it's good enough we may even pay hard cash for it too! So make sure you don't miss our STOS 3D tutorials – they could earn you some money.

STOS 3D is an extension to STOS Basic adding over 40 new commands to manipulate and create 3D environments created by you.

With STOS 3D installed, you can let your imagination run wild and create

games and demos featuring stunning 3D graphics as found in programs such as Cybercon III and Starglider II.

To help you design and create 3D objects, STOS 3D features a very powerful and easy to use Object Modeller



STOS 3D's versatile and powerful Object Modeller makes creating 3D objects quick and easy

GMAP

A utility that examines the level of fragmentation on both hard and floppy drives and displays the results as a graphical map

File fragmentation is a phenomenon whereby a file has been stored on disk in such a way that it has been split up into various parts rather than being continuous.

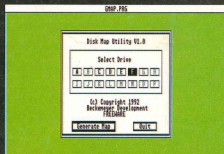
Heavy fragmentation is often responsible for poor performance in the amount of time a file takes to load and if that file is accessed frequently, the problem can become more frustrating.

GMAP provides you with a method of detecting whether poor disk performance is due to heavy fragmentation by presenting you with a graphical map of the contents and condition of your disk. The map consists of symbols representing continuous data, fragmented data, free space and also bad sectors.

When run, the program asks which drive you wish to check. Once the drive is chosen, click on Generate Map and the map is displayed. GMAP also displays the results and recommends whether optimisation is required.

Optimisation is the process whereby all the data is reorganised so that the data for each file is continuous and unfortunately, GMAP does not have a built-in optimiser but there are many available from PD libraries that do the job.

Programmed by: Beckemeyer Development
Configuration: All STs – med/hi-res
Filename: X_GMAP.TOS – Self-extracting archive



Selecting a drive to check is as easy as clicking a button

fully-working version of the powerful STOS 3D package! Read on...

3D

(OM). The OM includes all the tools and facilities you will need to create 3D objects as detailed as your imagination allows.

There is also the powerful facility to "paste" bitmap images directly onto the surface of your 3D object to add even more detail.

Imagine your 3D ship with your name written on the side of it as it flies through 3D space, or you could paste a bitmap design onto a ship as a missile hits the hull, thereby representing the damage inflicted.

By now you're probably raring to go, so follow the "Extracting STOS 3D" directions and then turn to our "STOS 3D Basics" on page 49 for details on installing STOS 3D onto your STOS Basic disks.

We will also be running a STOS 3D tutorial (starting in this very issue) in future issues to help you get the most from STOS 3D.

Extracting STOS 3D

STOS 3D needs to be placed on two disks called DEMO DISK and INSTALL DISK. Because of the large amounts of data that comprise each disk, the data for each had to be split up into two sets of related files to fit onto our CoverDisk.

These files are called X_DEMO1.TOS and X_DEMO2.TOS for the DEMO DISK and X_INSL1.TOS and X_INSL2.TOS for the INSTALL DISK. Follow the directions below for a successful extraction process.

1. Back up your CoverDisk! Always use a copy of the disk.
2. Format four disks and label them DEMO DISK, INSTALL DISK, WORK DISK 1 and WORK DISK 2 (refer to your Atari manual for details on formatting disks).
3. Insert your (backed up) CoverDisk and copy the files X_DEMO1.TOS onto WORK DISK 1 (refer to your Atari manual for details on copying files).
4. Now copy X_DEMO2.TOS onto WORK DISK 2.
5. Insert WORK DISK 1 into the drive.
6. Double click on X_DEMO1.TOS. The program will now begin to extract.
7. Once the extraction has finished, delete the file X_DEMO1.TOS from WORK DISK 1.
8. Insert WORK DISK 2 into drive.
9. Double click on X_DEMO2.TOS. The program will begin to extract.
10. Once the extraction has finished, delete the file X_DEMO2.TOS from WORK DISK 2.
11. Copy all files from WORK DISK 1 and WORK DISK 2 onto

the DEMO DISK.

12. Once files have been copied, delete all files on WORK DISK 1 and WORK DISK 2.

You should now have all the required files for the DEMO DISK. You now need to repeat the above steps in order to extract X_INSL1.TOS AND X_INSL2.TOS onto the INSTALL DISK, using the (now blank) WORK DISKS 1 and 2 again.

If all goes well, you should end up with two disks containing the following files on each:

DEMO DISK
 AUTO (folder)
 EXAMPLES (folder)
 OM (folder)
 UTILS (folder)
 OM.PRGM (Object Modeller. Double click to run)
 SIMPATHS (file)
 INSTALL DISK
 AUTO (folder)
 DEMOS (folder)
 C3D.PRGM (executable file)
 DEDME (documentation file)
 UPC.PRGM (executable file)
 UPL.PRGM (executable file)

Make a backup of these disks and put them in a safe place so that you will not have to go through the extraction procedure again should they become damaged.

Cold Revenge "Wrong Resolution" message

It seems that some people are experiencing a confusing error message when trying to play Cold Revenge, the multi-computer game that appeared on November's disk.

Apparently, after selecting one-player mode and choosing ships, clicking on DONE displays the error message "Wrong Resolution", even though the game was booted in the correct resolution.

The problem is that a incorrect configuration of ships has been selected, but instead of getting an error message that says incorrect configuration or something similar, you get the wrong resolution error, which is obviously confusing.

To get around the problem, you have to make sure you choose the correct ship configuration. As an example, consider a user wishing to play a one-player game with the user controlling one Federation ship against one Klingon ship, the procedure for choosing ships will be as follows:

1. Click on CA 1 to choose a Federation ship. The words COMPUTER LEADER 1 appear to the right.
2. Click on the words COMPUTER LEADER 1 to change it to PLAYER LEADER 1, indicating the ship is to be controlled by the user (this is the step most people seem to miss).
3. Now click on a Klingon ship (D7 1). The words COMPUTER LEADER 1 appear to the right. This indicates that this ship is to be controlled by the computer, so leave it as it is.
4. Click on DONE.

The game should then load normally with no sign of that annoying error message. Hopefully, if the author decides to develop the game further, a more user friendly selection process will be included.

ORDER FORM

Buy the STOS 3D manual for only £7.50

Get the most from our CoverDisk giveaway with this excellent user guide detailing all the STOS 3D commands with descriptions and examples showing how to use them in your STOS Basic programs.

Detach this order form and send it to Europress Direct, PO Box 2, Ellesmere Port, South Wirral L65 3EA.

I want to buy the STOS 3D manual for £7.50 Code No.9207

I am paying by
 Cheque/Postal Order/Access/Visa

Expiry Date

Card No.

Name:

Address:

Postcode:

Daytime Telephone No.:

☐ Please tick this box if you do not wish to receive promotional material from other companies

Please allow 28 days for delivery

STOS 3D basics

At one time, producing a good quality game incorporating 3D graphics meant having to master the complexities of matrix arithmetic and also having a good understanding of machine code programming.

Now, with the combined help of STOS Basic and STOS 3D, the ability to produce good quality 3D games is at the fingertips of everyone with a sound understanding of STOS Basic programming.

STOS 3D provides all the necessary tools for creating 3D objects as well as taking away the need to understand the mathematics behind all the calculations required to produce 3D graphics.

STOS 3D installs all these 3D commands to manipulate 3D objects into STOS Basic which can be easily called from your program.

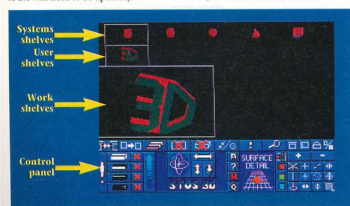
As soon as you have extracted the STOS 3D files from our CoverDisk (refer to Disk pages for details on extracting STOS 3D) you should have two disks marked Install and Demo. The first thing you should do is to make a backup of these disks.

Working copies

It is also wise to backup your STOS Basic disk and Compiler if you have them, as STOS 3D will require working copies of these to install itself onto.

To run the install program, simply put the Install disk into the drive and reboot your ST. The installation program should then automatically load and you will be presented with the installation menu screen showing three selections - READ ME, UPDATE and INSTALL.

Clicking on READ ME will give a brief explanation of the 3D installation disk. Clicking on UPDATE will allow you to update your current version of STOS Basic and Compiler to the latest version (STOS Basic and Compiler versions prior to 2.6 will need to be updated).



Darren Evans guides you through the installation and the basics of using this months amazing giveaway - STOS 3D



The Install Program main menu where you can Update STOS Basics and Compilers prior to version 2.6 and also Install STOS 3D

Getting to know STOS 3D

To help you to get the most out of STOS 3D, we will be running a tutorial over the coming months showing various techniques of programming with STOS 3D.

Also, there is a full ring-bound and comprehensive STOS 3D manual available which lists all the STOS 3D commands and their syntax as well as including comprehensive documentation on some of the more advanced commands and techniques.

Should you produce what you think is the best 3D game in the world, be sure to send it in to us! You may find your masterpiece appearing on our CoverDisk. So what are you waiting for? Get 3D-ing!

Clicking on INSTALL activates the installation procedure and places the 3D extension onto a copy of your STOS Basic and Compiler disks.

If you have a version of STOS prior to 2.6, now's the time to stop right here and follow the instructions on "Updating STOS Basic and Compiler".

STOS 3D should only be installed on v2.6 or higher. Those of you with v2.6 or higher should load up the Installation

program on the STOS 3D Install disk.

Once the main menu appears, click on INSTALL. You will then be prompted to place a backup of your STOS basic disk into Drive A. It is then just a matter

of following the on-screen instructions.

Once the 3D extensions have been installed on your STOS Basic disk, you will then be asked if you have the STOS Compiler. If you do, you will be asked to place your Compiler disk into drive A.

The heart and soul of STOS 3D is the Object Modeller (referred to as OM from now on). It provides you with a comprehensive and easy to use set of tools and utilities for designing and constructing all your 3D objects.

To run OM, locate the file OM.PRg and double click on it. OM will then load and an initial intro screen will appear. Click any key and you will be presented with OM's main display.

OM is quite a large program so if you're short of memory, try either running the OM from an AUTO folder or removing any accessories that you have in memory. OM uses an extra 32k when run from the ST's Desktop.

OM's main display is divided into two areas. The upper area is the display area and is a window into the 3D world where objects you create and assemble are displayed.

The display area is further subdivided into 12 shelves. Each of these shelves can hold one object. The system shelves contain the five basic shapes that constitute the building blocks for your objects.

At first, five basic building blocks may seem a little restrictive, but you are able to change the shape of each block and combine them to create completely different blocks. Below the five system shelves are the user shelves. These spaces provide you with a place for you to place work in progress, such as cus-

Updating STOS Basic and Compiler

You should only install STOS 3D on v2.6 or higher of STOS. If you have a version prior to this, there is a free update facility available from within the installation program (UPDATE) that comes with STOS 3D.

It is essential that older versions of both the STOS Basic disk and the Compiler are updated using this facility before selecting the install procedure.

If you do have a STOS version prior to 2.6, click on UPDATE from the main menu and follow the instructions displayed on the next screen.

The update process is completely automatic. If you have your STOS programs on a hard disk, you will have to update a floppy disk-based copy first, then copy the updated floppy disk version onto your hard disk.

After your STOS Basic disk has been updated, you will then be given the option of updating the STOS Compiler. If you have the STOS Compiler, type Y at the prompt and follow the instructions on screen.

Make sure you only update a backup of your STOS Basic disk first. Check that the updated version works, then update your original STOS Basic disk. Users of STOS Compiler should follow the same procedure.

tom blocks, half finished objects etc.

The remainder of the upper area is divided into two large work shelves. All of the modelling functions available to you from the control panel operate on these two areas.

The lower area is the control panel consisting of the many controls, tools and utilities that simplify the creation of your 3D worlds and objects. You may notice that except for the system blocks, the display area is blank and the actual shelves are not outlined or displayed in any way.

To see where the shelves are, or to select a shelf, simply hold the left mouse button down and move the mouse pointer around the screen. Each of the shelves will be highlighted as you move onto them. Familiarise yourself with the positions of each shelf.

Let's actually do something interesting now and manipulate one of the system blocks. One of the main functions you will use when creating objects is the Copy tool.

This allows you to copy the contents of one shelf to another and the procedure for doing this is as follows. Click the left mouse button on the shelf you want to copy from, then click the left button on the shelf you wish to copy to.

Now click on the Copy icon in the control panel and you will see that the object that was in the first shelf you selected is duplicated in the second shelf you selected.

Test this by copying the Cube system block into one of the two large work shelves. You will see that the Cube appears in the work shelf and is larger than the Cube shown in the system shelf. This is to facilitate easier and accurate manipulation when moving or joining objects.

To remove a object from a shelf, you use the Object Delete tool. To delete an object, click on the shelf containing the

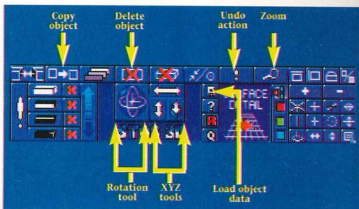
object you wish to delete, then click on the Delete icon. If you accidentally delete an object, you can click on the Undo icon to reverse the action. The Undo icon works for most functions that change the object in any way.

Once you have the Cube object in the work area, you are free to manipulate it in any way. To view the object from various angles and positions, you use the Rotation tool and the XYZ tools. This collection of icons lets you move and turn the object in the selected window to any position required.

To rotate an object, select the work area it occupies and hold the left mouse button down over the Rotation icon. Moving the mouse left or right will then rotate the object about the y-axis while an up/down movement rotates it about the x-axis.

The XYZ icons let you move the centre of a shelf in the direction indicated by the arrows. The left/right arrow icon moves the centre of the shelf in the x-axis, the up/down arrow moves the shelf in the y-axis and the last arrow moves the shelf towards and away from the observer.

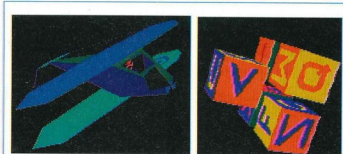
Each click of the left button on the arrow points moves the shelf a pre-set distance in that direction. Holding the



right mouse button down over any XYZ tool while moving the mouse will allow continuous movement using the mouse in that direction.

Other object manipulation icons allow you to stretch and re-size objects to form completely different shapes for building with. Such actions are performed by the Stretching tools.

These tools let you stretch an object both horizontally and vertically. To stretch an object horizontally, hold the left mouse button down over the horizontal stretch tool and move the mouse right to stretch and left to shrink.



Three examples of the kind of objects that you can create with STOS 3D and the Object Modeller

Backing up a disk

The following procedure is followed for backing up a disk using the ST's built in copying facility. A basic understanding of Desktop operation is required - if in doubt, refer to your ST's manual for instructions.

1. Boot up the ST's Desktop
2. Place a blank disk into drive A and format it using the sub-menu command for formatting from the file menu.
3. After formatting is complete, remove the blank disk and place the disk to be copied into drive A.
4. Drag the drive A icon onto the drive B icon.
5. Follow the on-screen instructions displayed.
6. Repeat instructions 2 to 5 for any other disks that require backing up.
7. Store the master disks that have been copied in a safe place.

STOS 3D commands

STOS 3D provides more than 30 new commands to your STOS Basic language, allowing easy manipulation of complex 3D objects. Here are just a few of them.

Td SCREEN HEIGHT	Sets 3D screen height
Td REDRAW	Redraw visible objects
Td CLS	Fast screen clear
Td DIR	List object files
Td LOAD	Load 3D object
Td OBJECT	Create 3D object
Td MOVE	Move an object
Td MOVE REL	Move object relative to current position
Td POSITION	Read object's world co-ordinates
Td ANGLE	Set object's attitude
Td ATTITUDE	Read object's attitude
Td BEARING	Calculate bearing and range of object
Td RANGE	Read range of object
Td COLLIDE	Detect a collision
Td WORLD	Convert local co-ordinates to world co-ordinates
Td SET ZONE	Define spherical zone around object

A whole new language

Once you step into the world of 3D graphics, you are exposed to a completely new set of buzzwords. Here are just a few examples of the 3D tech-speak that you will encounter.

Axes:	The three scales used to measure distances in a particular direction. The X axis runs left/right, the Y axis runs vertically and the Z axis points into the distance.
Co-ordinate system:	A set of axes usually at right angles to each other.
Local co-ordinates:	The co-ordinates based around a specific object.
Observer co-ordinates:	The co-ordinates based around the viewpoint.
Viewpoint:	The position in 3D space from which the view into the 3D world is seen.
Projection:	The result of moving a point or object to another co-ordinate system with a different number of dimensions.
Rotation:	Moving a point or object through an angle relative to a axis of rotation.
Translation:	A change in position.
Vertex/Vertices:	Any area on an object where lines meet at a point.

Once upon a time, in those long forgotten days of 1984, a classic 3D space simulation game was born on the BBC Micro. The basic idea was simple. All you had to do was travel from star to star, making deals, zapping the occasional enemy, and having a lot of fun in the process.

It sounded deceptively easy. But once you really got into the game, your mission became murderously addictive. You'd start playing casually, and find yourself entrapped in the thing for months!

Over the years, the game was successfully converted onto everything from the Sinclair Spectrum to the Atari ST. As computers evolved, the graphics improved dramatically, but the magic remained the same.

The only time the formula broke down was on the Amiga, where the conversion failed utterly due to sloppy programming. Typical!

Hopefully, you'll have guessed the name of the game. It was called *Elite*, and it's rightly regarded as one of the true classics in the history of computer games.

Buy it!

In the unlikely event that you don't actually own a copy, it's now available from Special Reserve for just £10.99. Rush out and buy it today! Believe me, you won't regret it.

Originality is rare in the computer industry, and best-selling computer games have a habit of spawning dozens of imitations.

After *Elite*'s phenomenal success, you'd naturally have expected a whole host of challengers. However, aside from fast action arcade games like *Warhead*, and feeble rip-offs such as *Federation of Free Traders*, there was practically nothing.

It's only now, eight years on, that we're finally promised a return visit, in the form of an official sequel from one of the original programmers.

Elite II has a hell of a tough act to follow, but the initial reports are looking decidedly promising. Suffice it to say, if it's anything like as good as its predecessor,

STOS 3D Tutorial

Effects which only a few years ago were unimaginable are now easy to achieve using STOS 3D. Stephen Hill's new series begins by tracking down that elusive third co-ordinate

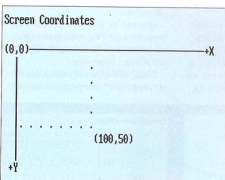


Figure II: Co-ordinates calculations can be simplified

or, we'll be in for a real treat!

The reason for the *Elite*'s isolation may seem puzzling, but it's probably understandable. The authors of *Elite* were tremendously gifted, and they achieved effects which seemed impossible on those early BBC Micros.

Elite was a massive game, and included over 2,000 planets in eight separate galaxies. Yet they packed the entire program into just 32k! That's about the size of a single ST screen.

Beating *Elite* was never going to be easy. So it's not surprising that most of the potential clones died still-born before they got off the drawing boards.

Nowadays, of course, everything's changed. The Atari ST represents a level of sophistication which would have been unimaginable in 1984. And it's finally possible to tread carefully in the footsteps of the original creators, and write our own 3D simulation games on the ST.

Unfortunately, in order to generate realistic 3D

effects, we'll have to push our ST to its absolute limits. This is going to be difficult even using ultra-fast 68000 assembly language, and unless you're a dedicated assembler freak, you might as well forget it.

If that applies to you, buy a book such as *Real Time 3D Graphics on the ST* by Andrew Tyler and program away. Alternatively, read on...

In an ideal world, we'd be able to write the whole game in a simple language like Basic. A few years ago, this would have been ludicrous, as Basic just wasn't up to the job. But that was before the introduction of STOS Basic on the ST.

Simple

STOS Basic took the skills of some of the best ST programmers, and concentrated their ideas into a range of simple, easy to use, commands.

Armed with STOS Basic, you could write anything from a fast arcade game to an adventure, without the need for complicated assembly language at all. It was incredible! Maybe I'm just a little biased, as I wrote the STOS Basic manual.

However, I could get better results in half an hour with STOS Basic than I could achieve in a day's worth of assembly language. I practically threw my assembler away.

Now, the same idea has been adapted for 3D. With the advent of STOS 3D from Europress software, 3D is no longer an

impossible dream, it's an achievable reality.

STOS 3D takes the original STOS Basic system, and adds in a host of powerful new 3D programming commands. These allow us to move, rotate, and animate our 3D objects directly on the screen, just like normal sprites.

Creating our objects is easy. We simply draw them up using the powerful Object Maker accessory, and load them straight into STOS Basic.

The entire 3D drawing system has been taken straight from a commercial game, and it's been written by some of the top programmers in the industry. So it's naturally capable of generating some really top-notch effects.

Suddenly, 3D Space Simulations are within the reach of all ST programmers. And we can finally allow our imaginations to run wild.

In the next few months, we'll be demonstrating this package in action. As an example, we'll have a bash at writing a small, but perfectly playable 3D space combat game using STOS 3D.

Routines

Along the way, we'll be developing general routines to handle 3D targeting, collision detection, laser beams, and real-time 3D combat. Once the game's complete, we'll then be able to expand it still further to create full-blown trading simulations, RPGs or 3D adventures.

Before we can flip our spaceship through our 3D universe though, we'll need to understand a bit about co-ordinates. These will allow us to keep track of the exact positions of all the various objects in our game, and move the spaceship realistically through our simulation.

All co-ordinate systems work in the same general way. They specify the position of an object using a list of distances taken from a fixed reference point known as the "origin".

On a map, the origin is a point on the Equator, and the distances are entered in the directions East, West, North, or South. So we could locate our object using a grid reference such as: (35

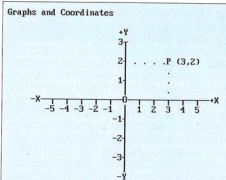


Figure I: Point at co-ordinates 3,2

A Three-Dimensional Cube

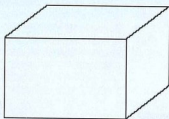


Figure III: Viewpoint facilitates 3D

East, 23 North) If we're drawing a graph, we normally place the origin at the centre, and read our distances along axes called X and Y. X measures the horizontal distance from left to right.

As a rule, values to the right of the origin are represented by a positive number, and values to the left are considered negative. Similarly, the Y axis enters the vertical distance, and increases as we go upwards.

Co-ordinates

Figure 1 contains a single point with the co-ordinates (3,2). As you can see, we've entered our co-ordinates in the order X,Y. This standard mathematical notation, and it's shorthand for:

(3 units in the X direction,
2 units in the Y direction)

Note that there's nothing special about the directions we're using, or the position of the origin point. Co-ordinate systems actually come in a variety of flavours, and there's no "right" way of doing things.

When we're positioning our graphics using screen co-ordinates, it's more con-

venient to place the origin point at the top left corner of the display.

The Y co-ordinates are now measured downwards, and negative values represent a point above the origin rather than below.

This system corresponds nicely with the way the screen is organised in the ST's memory, and it simplifies our co-ordinate calculations considerably. You can see an example in Figure II.

Up until now, we've only been dealing with two directions. These are fine for flat surfaces such as screens, but they're incapable of handling objects in the real, 3D world. We can however, easily extend our system to cope with the

extra dimension.

Let's define a new axis called Z. We'll now be able to fix the position of any object in 3D space using three values for the X,Y, and Z co-ordinates.

Unfortunately, there's a snag. The magazine you are reading, and the screen of your TV set, are both more or less flat, and they're unable to display solid objects in all three dimensions.

How can we possibly represent a 3D image on a 2D TV screen? Surely we'd need some sort of holographic 3D TV set? It seems impossible.

Luckily for us, there's a trick, known as "perspective". Perspective is basically

just common sense, and exploits the fact that real objects seem smaller as they recede into the distance.

Try taking this magazine, and moving it back and forth from your eyes. As you move it away, it will seem to get smaller, and as you zoom in, it will expand to fill your entire field of vision.

Now have a look at the image in Figure III. Although it's actually nothing more than a jumble of lines, your brain will cleverly assemble it into a solid-looking 3D cube.

Viewpoint

Notice how we've positioned the viewpoint slightly to one side. This allows us to see all three dimensions at the same time. If we viewed the cube head on, we'd only be able to see a single face, and we'd be left with a simple square.

Well, that's about it for this month. In our next exciting instalment, we'll be demonstrating some of these ideas in action.

So get your joystick ready, put on your flight helmet, and prepare yourself for a thrilling flight through the exciting world of STOS 3D.



Positioning an object using 3D coordinates

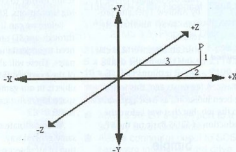


Figure IV: Representing three axes

Local Coordinates

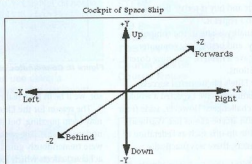


Figure V: Axes are relevant to current movement

Viewed from a different perspective

We can easily represent all three axes on our displays. The result can be seen in Figure IV. The Z axis now appears to go downwards through the paper. Note that the angle between the Z axis and the other axes is the same as the angle between X and Y, but it's squashed due to the perspective effect.

Take the point P as an example. It's positioned three units right, one unit up, and two units forward from the origin. So it has the co-ordinates (3,1,2).

Thankfully, STOS 3D handles the entire perspective system for us automatically. All we have to do is choose the co-ordinates of our objects and specify the position of our viewpoint. STOS 3D will now perform the calculations, and display the appropriate images on the screen.

The only minor complication is that STOS 3D uses two different "types" of co-ordinates. One set, known as the World co-ordinates, stores the positions of our

objects from a fixed origin placed at the centre of our universe. Each object in our game can now be fixed in place using its own set of three co-ordinate values. If we change these values, our objects will jump immediately to a new location.

Object zero is special, as it's used to represent the position of our viewpoint on the scene. Whenever it's moved, our spaceship will move accordingly, and we'll be left with the illusion that it's rushing realistically through our 3D universe.

World co-ordinates are great for measuring absolute positions, but they're lousy for things like targeting. As far as the player is concerned, the only distances that matter are the distances between the attacking objects and the actual ship. Anything else is irrelevant.

STOS 3D therefore allows us to measure our co-ordinates relative to any object we like. This has a lot

of advantages. If we read them from the current viewpoint, we'll be able to instantly discover the position of our attackers relative to the spaceship.

And if we measure them from the attacker, we'll be able to see things from the enemy's point of view as well.

These Local co-ordinates place the origin at the precise centre of our selected object. As you can see from Figure V, the axes are all relative to the current movement direction. So the X axis stretches from left to right, the Y axis goes from top to bottom, and the Z axis points forwards into the screen.

It's important to emphasise that the orientation of our local axes depends entirely on the orientation of the object we're using as the reference point.

Although we still refer to the directions as X,Y, and Z, they're actually completely different from the fixed axes we've provided for our World co-ordinates.

Last month, we saw how we could position our objects in a complex 3D universe using a set of three simple co-ordinate values. We'll now expand some of these ideas, and show how they can be applied in a real situation, using the amazing STOS 3D programming system.

Before we get stuck in, it's worth saying a few words about the STOS 3D package as a whole. STOS 3D is supplied on two 720k disks. The Installation program takes your original STOS Language disk and adds in 30 extra commands especially for 3D work.

Upgrading is easy. Providing you follow the instructions carefully, you should be up and running in a matter of minutes. You'll now be able to boot up your revamped STOS Language disk and access the new features directly from your Basic programs. All the new commands start with the letters TD (for Three Dee).

Let's progress to some serious 3D programming! But where do we actually start? Well, the obvious first step is to load some object definitions into memory. Object definitions are rather like the sprite images found in normal programs. Once they've been set up, they can be re-used by several different "objects" in our games.

The command we'll need is TD LOAD. Its syntax is just:

```
TD LOAD "filename"
```

where "filename" is the name of a definition file we've created using the object modeller.

```
TD LOAD "Ship1": REM Load in a Ship
```

Usually, STOS 3D expects to find all our definitions in the Objects directory of the current disk. This can be changed with the TD DIR instruction like so:

```
TD DIR "pathname"
```

Useful directories include:

```
TD DIR "A:\DEMOS\OBJECTS": REM For the
demo objects on the 3D Installation disk
TD DIR "C:\EXAMPLES": REM Accesses all
the examples from the OM disk
```

After we've loaded our definitions into memory, we'll need to assign them to a particular object "instance". These are similar to the Sprites used in an ordinary program.

Each instance is represented by a simple number from 1 to 20. So there's a maximum of 20 objects at any one time.

We can initialise our object instances using the TD OBJECT command. TD OBJECT is the 3D equivalent of the familiar STOS SPRITE instruction. However, since there are three dimensions, we'll need to enter a lot

STOS 3D Made Easy

It's easy to put STOS 3D's powerful features to real use. Stephen Hill shows you how to convert theory into practice

more information!

```
TD OBJECT n,"name",x,y,z,a,b,c
```

n is the object number from 1 to 20. Object zero is predefined, and is used to represent the player's viewpoint on the world.

x,y,z are the three "world" co-ordinates used to position our new object in 3D space. All co-ordinates are measured

from the centre of the object to the centre of our 3D world, in special units called VLU's (Voodoo Length Units).

Allowable co-ordinate values range from -8,000,000 to +8,000,000. As you can see from Figure 1, our game is effectively set in a large cube, with each side being around 16,000,000 VLU's in length. Just to give you some idea of the scale, we've included a few comparisons in Figure 11. The final three values enter

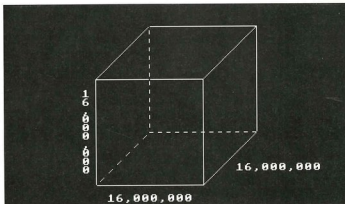


Figure 2: The Game World

The new commands

```
TD LOAD      load an object definition from the disk
TD OBJECT    assigns an object definition to an object instance
TD REDRAW    draws the object instances on the screen
TD MOVE      moves an instance to an new position
TD ANGLE      rotates an instance
TD INIT      reserves memory for a 3D program in bank 15
TD CLS       clears the working area of our 3D display
```

the orientation of the object, using three angles centred on the x,y, and z axes respectively. You can see how this works from Figure 11.

The axes serve as an invisible "spoke" around which the objects are allowed to rotate. The angles are specified in VRUs (Voodoo Rotation Units). They range from 0 to 65,535, and are measured in an anti-clockwise direction.

We can think of these angles in terms of a clock, with the face at the origin point, and the axes stretching outwards through the centre. Have a look at the diagram in Figure 14.

Our angle starts off from the three o'clock position. As it increases, it passes through positions 16,384 (12 o'clock), 32,768 (9 o'clock) and 49,152 (6 o'clock). And when it reaches 65,536, it jumps back to zero, and restarts from 3 o'clock.

Incidentally, when we save an object from the Object Modeller, the current orientation is saved as well. So it's vital to make sure that the object is facing in a sensible direction before storing it on the disk.

Take the following example:

```
TD OBJECT
1,"ship1",1000,2000,3000,16384,32768,49152
```

This defines object 1 using a shape called "ship1" at co-ordinates 1000,2000,3000. The attitude is set to 16,384, 32,768 and 49,152 (or 90,180, and 270 degrees).

Note that TD OBJECT only assigns our instances to a definition. It doesn't actually display them on the screen. That's the job of the separate TD REDRAW command.

```
TD REDRAW screen address
```

TD REDRAW redraws all our instances in one smooth burst, taking the viewpoint from the current position of object zero. Initially, the viewpoint is set to co-ordinates 0,0,0, but it can be effortlessly moved around using the TD MOVE or TD FORWARD commands.

Screen address is the address in memory of the STOS Basic screen which will be used for our display. This could be either the Physical or Logical screens, or it might be a new screen we've created with RESERVE AS SCREEN.

Try:

```
TD REDRAW Logic: REM Draws the objects on
the logical screen
```

Or:

```
TD REDRAW Physic: REM Draws the objects on
the physical screen
```

Example A shows these instructions in action. Here's a small example of these instructions working, "game.sh1" is an object representing a simple space-ship. It's taken from the 3D Installation disk. The Amos version was actually called

Figure 11: A sense of scale

► "game_ship", but Europress sadly ran out of characters!

TD INIT reserves 100,000 bytes for our objects in memory. The format is simply:

TO INET size

TD INIT should always be the first TD command in our program, and it can only be used once. The amount of space we'll require depends on the number and types of the objects we've defined in our routines. The default is about 50k.

Now we've got a simple space-ship, we can move it around with the powerful TD MOVE instruction.

TO MOVE D-X-Y-Z

Example 2 demonstrates this command. It slowly moves object number one towards our current viewpoint (0,0,0). Unfortunately, the ship flickers uncontrollably. That's because we are continually erasing it and redrawing it from the display. Yuck!

Although the process sounds highly complicated, the whole thing can be accomplished using just a couple of lines:

Screen Swap swaps between the logical and physical screens, and neatly flicks the completed image onto the display. After the program has concluded,

TO ANGLE $n.a.b.c$

Try adding the following lines to Example B:

Here are some alternative versions:

Powerful

The second disk contains the Object M short. This is a powerful drawing program with everything we need to create attractive

Don't attempt to use the **Object Modeller** straight from the box. It's a recipe for instant frustration! Before you do any experimenting, you're strongly recommended to work through the tutorial on page 11 of the **STOS 3D** manual. You'll save yourself a lot of grief!

```

10 rem Boring initialisation stuff
20 key off : mode 0 : hide on : curs off
30 ts to int: 100000
40 key on : curs on
50 input "Insert copy of the 3D installation disk into drive A: and press Return";AS
60 dir "A:\DEMOS\OBJECTS"
70 to load "game_shi"
80 rem initialise objects
90 palette 0,0,0,0,0,0,0,0,0,7777,8555,8226,8761,8744,8622,8500, 8727
100 to object 1,"game_shi",0,0,2000,90*182,0,0
110 to redraw logic
120 wait key
130 show on : curs on

```

Next month, we'll be using these commands to generate a working 3D space simulation. In the meantime, have a bash at using TD MOVE and TD

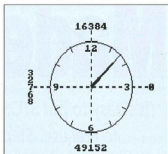


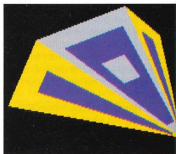
Figure IV Angles and VRUs

```

10 rem Boring initialisation stuff
20 curs off : key off : hde : mode D
30 rem load 1000000
40 rem copy object definition for ship
50 input "Insert code of the 3D installation disk into drive A: and
press Return":AS
60 dir dir:"A:\DEMOS\OBJECTS\"
70 rem load "game.shi"
80 rem rem initialise objects
90 palette 0,0,0,0,0,0,0,0,$777,$555,$222,$767,$744,$622,$500,$727
100 rem obj type 1,"game_shi",0,0,$0000,0,0,0
110 rem Main game loop
120 logic-chase
130 for $I$T=$000 to 660 step-10
140 do move 1,0,0,$I$T
150 ANG=$I$T*100 mod 45536
160 do angle 1,ANG,ANG,ANG
170 rem cls logic
180 rem cls logic
190 do redraw logic
200 screen swap
210 next $I$T
215 rem The next bits not included in the article
220 screen copy physic to logic
230 logic-physic
240 curs on : show

```

Example B



A close look at the ship

The second disk contains the Object Modeller, or OM for short. This is a powerful drawing program which provides us with everything we need to create attractive objects for our games.

start-up disk, and expects it to be permanently write-enabled. So if you overwrite one of the example definitions by mistake, it could be lost forever.

Begin by creating simple shapes such as pyramids and cubes. These are often ideal for your games, and it's pointless to muck around with the advanced features until you've completely mastered the basics.

As a rule, the simpler the object, the quicker it can be displayed on the screen. Bear this in mind when you're designing fast moving objects such as missiles or spaceships.

Finally, check out the example objects from the **Example** folder of the **OM** disk. These provide an impressive demonstration of the entire system in action.

Hopefully, you'll now be well on your way to becoming a expert OM user.

Until now, all our examples have been restricted to simple demos. This month, we'll be progressing to a full blown space simulation!

We'll begin by defining a few objectives. What we are aiming for is a movable "viewpoint" which can be controlled directly by the joystick or mouse. You can see the idea from Figure 1.

In order to meet these objectives, we'll have to solve three main problems.

1. How can we read the joystick, mouse, and keyboard for STOS Basic?
2. How can we move the viewpoint through our 3D game world?
3. How can we steer our spaceship in a believable way?

Once we've found answers to these questions, we'll be well on our way to creating the finished program.

The first bit is easy, as STOS Basic provides us with a vast range of commands for reading the various controls. The keyboard, for instance, can be checked using the INKEY\$ and SCANCODE functions.

The INKEY\$ function just reads a keypress from the user, and places the appropriate character into a string variable. Unlike the familiar INPUT command, INKEY\$ doesn't display the new character on the screen and it doesn't wait for a keypress.

So we can check the keyboard at any time, without interfering with the rest of our program.

```
$(INKEY$;REM Load a keypress into variable C
```

\$ INKEY\$ is great for entering letters or numbers, but it only works with keys which have an equivalent ASCII value. Some keys, such as Help, and the cursor arrows, require special treatment, and

STOS 3D Made Easy

Stephen Hill continues his guide to this powerful but simple programming package

should be checked using a separate SCANCODE function instead.

SCANCODE returns the "scancode" of the last key which was entered using INKEY\$. This code is a simple number, ranging from 0 to 255. Here's a crude test program which will allow you to try out the various keys:

```
10 key off:scans off:cl:off: mode 0
20 print "Hit a key"
30 $(SCANCODE;SCANCODE
40 if $(SCANCODE="" then print "Character:";$(SCANCODE)
50 if $(SCANCODE=0 then print "Scancode:";SCANCODE
60 goto 30
```

We'll now have a brief look at the STOS Basic joystick commands. There are four key functions we'll need to

examine:

```
=LEFT - checks if the joystick has been dragged left
=RIGHT - checks if the joystick has been pulled right
=UP - tests for an upward push of the joystick
=DOWN - tests for a downward pull
```

These functions return a value of either -1 (TRUE) or 0 (FALSE). If we get a value of -1, then the test has been successful, and the joystick has been moved in the selected direction. If not, we'll get a zero instead.

Try out the following example:

```
10 key off:scans off:hide:mode 0
20 centre "Pull the joystick"
30 if left then locate 8,12:print "Left"
40 if right then locate 26,12:print "Right"
50 if up then locate 19,8:print "Up"
60 if down then locate 18,16:print "Down"
70 goto 30
```

The final control option is via the mouse. This can be tested using the X MOUSE and Y MOUSE functions:

```
=X MOUSE - returns or changes the X co-ordinate of the mouse pointer
=Y MOUSE - reads or modifies its Y co-ordinate
```

On the face of it, these functions seem pretty useless, as they only return the physical co-ordinates of the mouse pointer on the screen.

However, with a bit of cunning, we can easily trick them into returning a

relative movement rather than an absolute screen position. This data can then be used to control our spaceship, in a similar way to the previous joystick commands.

Our first job is to remove the mouse pointer from the screen using HIDE, so that it won't interfere with our game:

```
10 Hide
```

Next, we initialise the mouse by moving it to the centre of the screen using the X MOUSE and Y MOUSE functions:

```
20 X Mouse=160:Y Mouse=100
```

We can now read our mouse with:

```
30 XC=X Mouse-160:YC=Y Mouse-100
```

Line 30 works out the distance the mouse has been moved from the centre point, and loads the results into the variables XC, and YC.

Finally, we replace the mouse back at its original position like so:

```
40 X Mouse=160:Y Mouse=100
```

Here's the finished demo:

```
10 cls:key off:hide
20 DAMP=15:x mouse=160:y mouse=100:rem 1'll be explaining DAMP in a moment
30 XM=x mouse-160:YM=y mouse-100
40 x mouse=160:y mouse=100
50 XC=XM/2:YC=YM/2
60 rem if XM then XC=C+sgn(XM)*DAMP:YC=Y+sgn(YM)*DAMP
70 if XC/2 then print "Mouse has moved right"
80 if YC/2 then print "Mouse has moved down"
110 rem XC=X/2:YC=Y/2
120 wait 10
130 goto 30
```

This routine requires a separate copy of the mouse for each and any every response. If you'd prefer to use an "inertial" system, you'll find the appropriate code in the REM statements in lines 60 and 110. Simply remove line 50, and add the following lines:

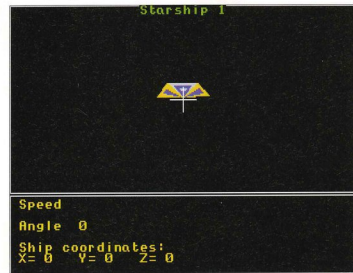
```
60 if XM then XC=X+sgn(XM)*DAMP:YC=Y+sgn(YM)*DAMP
110 XC=X/2:YC=Y/2
```

These instructions remember the present control setting until the user changes direction. Line 110 sets up a damping factor which stops the mouse in its tracks after a few seconds. The higher the value of DAMP, the longer the effect will last.

We're now in a position to tackle the second part of our problem. How can we move our cockpit realistically through our 3D world?

Well, if you've been following the series carefully, you'll remember that STOS 3D draws all our objects from a specific viewpoint assigned to object zero.

So all we have to do is move object zero, and the cockpit will automatically



Looking out of the cockpit, the ship object is some distance from us and directly ahead

Starship 1



Speed
 Angle 0
 Ship coordinates:
 X= 0 Y= 6 Z= 4057

Pressing the + key increases our forward speed and as a result, the ship object draws closer

move in step.

One obvious approach would be to move the viewpoint directly using TD MOVE. This would lead to a line such as:

```
500 Td Move 0,VX,VY,VZ
```

Where VX, VY, and VZ are assumed to hold the new world co-ordinates of our spaceship. We could now change these co-ordinates using any one of the control systems we've just been examining.

While there's nothing inherently wrong with this system, it's worth noting that there are several alternative options:

```
TD MOVE REL n,dx,dy,dz
```

This command moves an object relative to its present location. The new world co-ordinates will be immediately increased by DX,DY,DZ units respectively.

```
500 Td Move Rel 0,-100,50,200
```

moves the viewpoint 100 units left, 50 units up, and 200 units forward from the original position.

There's also the TD FORWARD command we looked at earlier:

```
TD FORWARD n,dist
```

TD FORWARD moves our object DIST units forward in the current direction:

```
500 Td Forward 0,SPEED:ren Moves the ship  
SPEED units forward
```

The speed can be controlled in a number of ways, depending the game. For now, we'll allow the user to increase or decrease the speed directly from the keyboard.

```
240 CS=inkey:is=scancode  
250 if CS="+" and SPEED=0 then dec SPEED
```

With the aid of TD MOVE REL, or TD FORWARD, the ship can now be moved at will. However, we've still not decided on the direction!

There are lots of possible approaches we can take to this problem, but it's easiest to use the following procedure.

Assign one control for changing the angle of the cockpit. Depending on the angle we've chosen, the ship will appear to turn, dive, or twist in space.

```
500 Ren rotate ship around the X axis  
generating a dive/climb effect  
510 if JUP then LA=LA+182: Td Angle 0,LA,0,0  
520 if JDOWN then LA=LA-182: Td Angle 0,LA,0,0  
530 return  
  
600 Ren rotate ship around the Y axis,  
turning the cockpit left or right  
610 if JLEFT then LB=LB+182: Td Angle 0,0,LB,0  
620 if JDOWN then LB=LB-182: Td Angle 0,0,LB,0  
630 return
```

These modules rotate the cockpit by 182 VRUs (approx 1 degree).

```
10 ren Starship 1  
20 ren Initialisation  
30 curs offkey:offhidemode:Doclick  
offscreen 13,10000  
40 set mark 2,30:ren Try marks 3 to 6  
50 td init 100000  
60 ren load up object definitions  
70 input "Insert copy of the 3D  
installation disc into drive A: and press  
Return":$A  
80 td dir "A:\DEMOS\OBJECTS"  
90 td load "game_shi"  
100 td load "dice"  
110 ren initialise objects  
120 palette  
130,4777,4700,470,47,4770,477,4707,4777,455  
5,4226,4761,4744,4622,4500,4727  
140 td init "game_shi",0,0,5000,0,0,0  
5000,0,162490,0  
150 logic:back  
160 ren Main loop
```

Here's a completed example for you to play around with. Notice how we've positioned the status display just below the drawing area used by STOS 3D. The size of this area can be changed by calling the TD SCREEN HEIGHT command near the start of your program

Starship 1



Speed
 Angle 354
 Ship coordinates:
 X= 464 Y=-9 Z= 3637

Moving the joystick alters the view from the cockpit and the ship object changes position

Next, we use the second control to shift the position of the ship. The direction of this movement will naturally depend on the angle we've previously chosen to rotate.

So if the first control generates a dive, the second should move the ship horizontally. And if we're already able to turn the ship, we'll need some way of shifting the cockpit up or down:

```
700 Ren Move ship Left. Use if JUP and  
JDOWN handle dives or climbs  
710 if JRIGHT then td move rel 0,100,0,0  
720 if JLEFT then td move rel 0,-100,0,0  
730 return  
  
800 Ren Move ship up/down. Use if JLEFT and  
JRIGHT turns the cockpit sideways  
810 if JUP then td move rel 0,0,100,0 420  
14 down then td move rel 0,-100,0 430  
return
```

The advantage of this technique is that it's both fast, and simple to program. If we tried to turn the cockpit in any direction, like Elite, we'd be faced with lots of extra complications, which

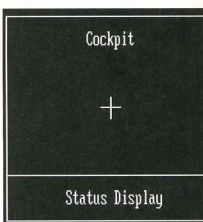


Figure 1 - the movable viewpoint

would dramatically slow down our game.

● Next month, we'll be taking our new ship, and adding in the laser and the targeting systems. See you then!

```
170 cls logic:ren Clear screen  
180 $IDE=0:ren Set horizontal flight flag  
190 ren draw joystick  
200 if JRIGHT then td move rel 0,100,0,0:$IDE=1  
210 if JLEFT then td move rel 0,-100,0,0:$IDE=1  
215 ren MOD cycles LA from 0 to 65535  
220 10 JUP then LA=(LA+182%65535) mod 65535  
230 14 JDOWN then LA=(LA-182%65535) mod 65535  
240 CS=inkey:is=scancode:ren read keyboard  
250 if CS="+" and SPEED=0 then  
SPEED=SPEED+4  
270 if CS="-" then SPEED=0  
280 ren check help key  
290 if S=9 then td move 0,0,0,0:td angle 0,0,0,0:SPEED=0:LA=0  
300 clear key:ren Clear existing keypresses  
310 ren move ship forward  
320 if $IDE=0 then td forward 0,SPEED  
330 td angle 0,LA,0,0:ren handle dive  
340 ren sound effects
```

```
350 volume SPEED/3:if SPEED=0 then noise  
20  
360 td redraw logic:ren draw 3D  
370 ren display cockpit on top of 3D  
380 ink 1:box 0,0 to 319,139:pen 3:locate 0,0:centre "Starship 1":pen 5  
390 polyark 160,70:box 0,141 to 319,199  
400 locate 1,18:print "Speed":if SPEED=0  
then ink 3:bar 64,144 to SPEED+64,152  
410 locate 1,20:print "Angle"  
"LA/182:locate 1,22:print "Ship  
coordinates:"  
415 Ren TD POSITION reads the coordinates  
of an object. See next month  
420 locate 1,23:print "X":td position  
x(0):" Y":td position y(0):" Z":td  
position z(0)  
430 screen swap:wait vbl:ren flick  
display onto screen  
440 goto 160
```


STOS 3D Made Easy

Now Stephen Hill adds a laser and targeting system to our rapidly developing STOS game

We'll start off with a simple scanner routine. Its job will be to take each object in our game-world, and plot its position on a neat little radar screen.

Our first objective will be to find the latest positions of all the nearby objects in our game. This can be accomplished with three TD POSITION functions:

```
x=TD POSITION X(object number) Reads
the X co-ordinate of an object.
y=TD POSITION Y(object number) Gets
the Y co-ordinate of our object.
z=TD POSITION Z(object number) Returns
its Z co-ordinate.
```

Once we've found the physical co-ordinates of these objects, we'll need to work out their actual distances from our spaceship. We can do this by converting all our co-ordinate values into their equivalent local versions.

STOS 3D supplies us with a direct way of converting any group of world co-ordinates directly into their local equivalents:

```
lx=TD VIEW X(object number,wx,wy,wz)
```

This takes a location in the 3D game world with co-ordinates wx,wy,wz, and returns the local X co-ordinate of this point as seen from our chosen object.

```
ly=TD VIEW Y(object number,wx,wy,wz)
```

Gets the local Y co-ordinate. That's the vertical distance from our object to the point at wx,wy,wz:

```
lz=TD VIEW Z(object number,wx,wy,wz)
```

Returns the local Z co-ordinate corresponding to the point wx,wy,wz.

The object number can be anything

we like, but since we are normally interested in the viewpoint, it's usually set to zero.

Each of these TD VIEW functions automatically works out all three of the required co-ordinate values in advance. So after we've entered the world co-ordinates into the first function, we can get an instant read-out of the others using a second, faster form of these commands:

```
ly=TD VIEW Y(O) (lz=TD VIEW Z(O))
```

The final stage is to reduce our distances into a form which can be plotted onto the screen. We'll therefore divide them by an appropriate scale value, so that they lie within the physical boundaries of our resulting scanner display.

Here's a line by line explanation of our finished routine. Don't type it in on its own though. It won't work! First, we

```
440 MRANGE=24000 : rem set maximum scanning range
450 XSCALE=MRANGE/CSIZE : rem Set X scale
460 ZSCALE=(MRANGE/CSIZE)*1 : rem Set Z scale
470 rem Main Scanning loop
475 rem NOBJECTS = the number of objects in the
    game
480 for OB=1 to NOBJECTS
490 RANGE=td range(O,OB) : rem Get distance
    between objects O and OB
500 if RANGE<MRANGE then gosub
    570 : rem Call the scanning routine.
510 next OB
560 rem Scanner routine
570 Xw=td position x(OB) : rem Get X Co-ordinate
    of object number OB
580 Yw=td position y(OB) : rem Get Y co-ordinate
    of OB
590 Zw=td position z(OB) : rem Get Z co-ordinate
600 XL=td view x(O,Xw,Yw,Zw) : rem Convert from
    WORLD to LOCAL co-ordinates
610 YL=td view y(O)
```

define a few constants at the start of our program:

```
42 rem OBCOL = Colour of object on the
    scanner
43 dim OBCOL(2) : OBCOL(1)=1 : OBCOL(2)=2
46 NOBJECTS=2 : Rem Enter the number of
    objects
47 CTX=160 : CTY=159 : CSIZE=40 : rem
    Set centre and width of scanning window
48 THEIGHT=140 : Rem Holds the height of
    the viewing window
```

Then, we display a rounded box at the centre of the status area:

```
419 rem New version of the co-ordinate
    display to make room for the scanner
420 locate 1,21 : print "X=";td position
    x(O) : locate 1,22 : print "Y=";td position
    y(O) : locate 1,23 : print "Z=";td position
    z(O)
421 rem The Scanner window
425 ink 0 : bar CTX-CSIZE,CTY-CSIZE to
    CTX+CSIZE,THEIGHT+1
430 ink 10 : rbox CTX-CSIZE,CTY-CSIZE to
    CTX+CSIZE,CTY+CSIZE-1 : draw CTX-
    CSIZE,CTY-1 to CTX+CSIZE,CTY-1 : rem
    Draw scanner
```

We can now set up the scanning range, scan our surroundings, and plot the results onto the display – see Listing 1.

Notice the TD RANGE command at line 490. This returns the straight line distance between any two objects in our game. It's used to limit our scanner display to the objects nearest our spaceship. The format is:

```
r=TD RANGE(object 1,object 2)
```

In our scanner, object 1 is the viewpoint, and object 2 represents our attackers. So the final line is just:

```
490 RANGE=td range(O,OB)
```

Let's start work immediately on our laser. We can produce this routine in three stages, starting with generation of the actual laser beams. Here's how this system will work in our game:

```
235 if fire or mouse key then BEAM=true
    : boom : wait 5 else BEAM=false
```

```
620 ZL=td view z(O)
621 rem Scale the values to between -CSIZE and
    +CSIZE
630 XP=XL/XSCALE : YP=ZL/ZSCALE
640 ink OBCOL(OB) : rem Set colour of current
    object
650 rem If the objects above the ship display an
    UP arrow
660 if YL>0 then draw XP+CTX-3,YP+CTY to
    XP+CTX,YP+CTY-3 : draw to XP+CTX+3,YP+CTY
670 rem If the objects below the ship display a
    down arrow
680 if YL<0 then draw XP+CTX-3,YP+CTY to
    XP+CTX,YP+CTY+3 : draw to XP+CTX+3,YP+CTY
690 rem If it's perfectly aligned display a
    horizontal line -
700 if YL=0 then draw XP+CTX-3,YP+CTY to
    XP+CTX+3,YP+CTY
705 rem Lines 710-820 are waiting for the
    targeting system used by the Laser
830 return
```

Listing 1

BEAM is a "flag" variable that will be used to tell the rest of our program at the player has fired the laser. After the laser has been discharged, we can display it on the screen using a couple of simple STOS Basic DRAW commands:

```
466 rem Fire laser
467 if BEAM then ink 3 : draw 0,140 to
160,75 : draw 160,75 to 319,140
```

These commands should be executed just after the TD REDRAW instruction, but before the final Screen Swap. Our laser beams will be automatically removed after the next turn of the drawing loop.

Detecting a successful hit is very easy, and can be discovered using a little-known STOS Basic Graphics function called Point:

```
cp=POINT(x,y)
```

Point just returns the number of the colour index x,y at the selected screen position. We can exploit it to load the colour of the point immediately under our sights:

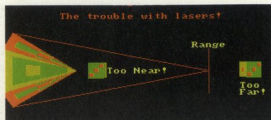
```
365 TESTCOL=point(160,75)
```

Since STOS 3D only uses colours from 8 to 14, we can now apply a simple test to this value to instantly detect whether our attack has been successful. See line 720 in the resulting program.

We then have to decide which object has been hit. If our spaceship were real, the laser beams would only meet at a fixed point in our 3D world. So unless the attacker was at precisely the right range, it could fly through our laser barrage with impunity. You can see the effect above.

In an actual game however, it's far more natural to ignore the range completely, and assume that any object that enters our sights will be fair game. The basic idea is to examine all our objects in turn, and subject them to a number of tests to see if they're directly in front of our sights.

We can begin by checking to see whether the object can actually be seen from the present position. If not, we can reject it immediately as a potential target:



Laser beams meeting at a point

```
v=TD VISIBLE(object number)
```

TD VISIBLE checks the status of our object on the screen, and returns a value of -1 (True) if it's visible, or 0 (False) if it's not displayed.

Once we've discovered that our object is visible, we can calculate its physical position on the screen using the TD SCREEN functions:

```
sx=TD SCREEN X(wx,wy,wz) sy=TD SCREEN
Y(wx,wy,wz)
```

These return the apparent screen position of any point in our 3D universe. If this point lies outside the current viewing area, TD SCREEN will return a value of -1 instead.

As with TD VIEW, we can omit the world co-ordinates completely after we've entered them into the initial TD SCREEN X command, for example:

```
SX=TD SCREEN X(wx,wy,wz) : SY=TD SCREEN
Y(O)
```

The required object co-ordinates can be taken directly from our existing scanning routine. They're available from the variables XW, YW and ZW:

```
710 rem Check if laser has hit a ship
720 if BEAM=false or TESTCOL<7 or td
visible(OB)=0 then return : rem NO
730 rem YES
740 rem get the screen co-ordinates of
the centre of our object
750 SX=td screen x(XW,YW,ZW) : SY=td
screen y(O)
760 if SX<0 or SY<0 then return : rem If
the object is out of range, return
```

After line 750, SX,SY contain the screen co-ordinates of the centre of our current object. We now can use these co-ordi-

notes to see whether this object is within our sights. We start off by working out the distance from the centre of the ship to the centre of our crosshairs:

```
770 DX=SX-160 : DY=SY-75 : rem Find the
gap between the object and the sights
780 D=sqr(DX*2+DY*2) : rem work out
straight Line distance using trigonome-
try
```

Variable D now holds the distance from the sight to the centre of our object. This distance can be subsequently manipulated to check whether our sights lie within the visible boundaries of the object on the screen.

Since the appearance of our object varies dramatically depending on our current viewpoint, we can't perform this test directly. It's far easier to wrap our object in an invisible sphere, centred at its present co-ordinates, and use that instead.

The object modeller calls this region a "circumsphere". Its radius is set to the minimum size required to enclose the object completely. We can find its dimensions by simply loading our object into the OM, and selecting the "R" icon:

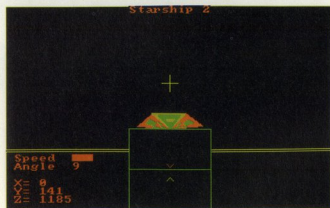
```
45 dim CIRCUM(2) : CIRCUM(1)=750 :
CIRCUM(2)=300
```

Circum is defined at the start of our program, and holds the initial sizes of the spherical test zone for each object in our game. Here's how we calculate the size of this zone:

```
790 gosub
850 : rem Find out apparent size of
object and place it in RD
800 rem If the Distance is less than RAD
and there's no nearer object
810 rem that already satisfies the test
conditions, then current OBJECT=TARGET
820 if B=RD and RANGE<LRANGE or
LB>LRAD then TARGET=OB : LRANGE=RANGE :
LB=0 : LRAD=LB
830 return
840 rem Find the visible radius of the
object on the screen
850 XH=td world x(O,XL-CIRCUM(OB),YL,ZL)
860 rem Get the co-ords of the point
CIRCUM(OB) units to the left of the
object
870 YH=td world y(O) : ZH=td world z(O)
880 HX=td screen x(XH,YH,ZH) : rem Find
X co-ordinate of this point on the
screen
890 RD=abs(SX-TX) : rem Work out an
approximate radius
900 if TX<0 then RD=CIRCUM(OB)
905 rem Arc SX,SY,RD,0,3600 : rem
(remove first rem to see the test zone)
910 return
```

Finally, we can display the chosen target from within our program:

```
520 ink 2
530 if BEAM and TARGET then locate 26,20
: print "You hit ";TARGET;" " : bell :
wait 5
540 screen swap : wait vbl : rem Flick
display onto screen
550 goto 160
```



Three TD functions return co-ordinates for x,y and z

STOS 3D Made Easy

Welcome to the fifth and penultimate instalment of my STOS 3D programming series. This month, we'll be developing a series of routines to flip our attackers through complex 3D manoeuvres.

Before we get too involved, it's worth doing a little careful groundwork. Our opponents can be divided up into several distinct classes.

Missiles will fire in a straight line in the direction we are currently facing. They'll continue until they either hit an object, or move out of range.

Guided missiles will track a selected target object through our 3D universe and there'll also be realistic spaceships. These will be piloted by an imaginary crew, with imaginary personalities, but with a very real killer instinct.

Each of these objects will need to be treated quite differently in our game. So we'll have to generate a separate program module for each possibility.

The best way of achieving this is to provide a general purpose combat routine, which can call up the appropriate strategy automatically.

Each strategy will be represented by a single number inside our SI, which will then be used by the combat module to select the required routine. Here's how this would look inside our Basic program:

```
rem Main combat module
On TACTICS Gosub
attack,defend,follow...
Return
```

Where TACTICS holds the number of the strategy we've selected for the current object, and attack, defend and follow are assumed to be the line numbers of our individual combat routines.

In a real game, we'd hold the tactics data in an array:

```
Dim
TACTICS(number_of_objects,no_of_items)
On TACTICS(object_number,0) Gosub
attack,defend,follow...
```

TACTICS(object_number,0) would now contain the strategy for each object in our game. The extra dimension would provide room for additional information as appropriate. A guided missile, for example, might have the object number of its intended target loaded into

Illegal aliens

To achieve convincing enemy movement requires some judicious planning. Stephen Hill continues our game by doing just that...

TACTICS(object_number,1), and its speed in TACTICS(object_number,2).

The great thing about this approach is that it's totally general. So once we've got it up and running, there's nothing stopping us from developing new attack formations, and new ship types at will.

Now that we've sketched out the basic technique, we can start to examine how it might be performed in STOS 3D.

STOS 3D contains a range of built-in facilities that make it really easy to generate terrific attack sequences for our games. The key, is to use a deceptively simple command called TD FACE.

TD FACE takes an object and turns it to face a selected point in our 3D game world, so

```
TD FACE object1,object2
```

turns object 1 so that it faces object2.

On the face of it, TD FACE doesn't sound so hot. But when it's used with TD FORWARD it allows us to create a mouth-watering range of effects.

Let's have an immediate bash at producing those guided missiles I mentioned a few moments ago.

The first job would be to select our target object using a variation of our previous targeting system. We could then move the missiles with the following commands:

```
Td Face MISSILE,TARGET : Rem Turn
missile towards target
Td Forward MISSILE,50 : Rem Move it 50
units forwards
```

These lines would be called up as part of scanning routine using the information we've previously placed in the TACTICS array. Every time they were performed, TD FACE would turn the missile towards the target's new position, and TD FORWARD would move it 50 units closer.

The target could now dodge and dive as much as it liked. The missile would follow it relentlessly.

So far, TD FACE has shown its worth

in the creation of simple missiles. But spaceships need to be moved through much more complicated manoeuvres.

For these, we'll use a second, more advanced version of the command.

```
TD FACE object,wx,wy,wz
```

turns our object so that it faces the point with world co-ordinates wx,wy,wz. As it stands, it's not too impressive, as world co-ordinates aren't all that useful. However, with the help of the TD WORLD functions, we can also use local co-ordinates, measured relative to any object in our game. This adds a whole new dimension to our capabilities.

Take the following simple scenario. Our player has fired off a guided missile at one of the aliens. And the imaginary pilot has detected this missile on his imaginary scanner, and has had an imaginary heart attack!

He's thought carefully about the missile, analysed its nearness to his ship and its explosive potential, and has logically decided on the optimum course of action. He's going to run away, terribly fast!

How might we duplicate this strategy in STOS Basic? Well, the pilot could be simulated in a simple loop. His first action, would be to turn his ship away from the missile. This might correspond to a point 10,000 units in front, say at local co-ordinates (0,0,10000).

Our program could now convert these co-ordinates into the equivalent World versions like so:

```
FX=Td World X(MISSILE,0,0,10000):FY=Td
World Y(X):FZ=Td World Z(X)
```

Variables FX,FY,FZ would then hold the world co-ordinates of the point 10,000 units in front of the missile. So the ship could be immediately turned towards this position using TD FACE.

```
Td Face ALIEN,FX,FY,FZ
```

Once the pilot had got the ship pointing towards safety, he'd step on the gas

pedal, and start accelerating.

Td Forward ALIEN,150

These commands would be repeated until either the alien or the missile was destroyed.

In a real game, the situation would obviously be more complicated, as the attacker would be continually changing course to evade the missile. We could simulate these manoeuvres by adding an extra random element to this routine. Instead of pointing the alien directly in front of the missile, we'd shift the aim horizontally or vertically.

```
XOFF=Rnd(10000)-5000: Rem Point alien
left/right by random amount
YOFF=Rnd(10000)-5000: Rem Point alien
up/down
ZOFF=10000: Rem Point the alien 10,000
units forwards
```

```
FX=Td World
X(MISSILE,XOFF,YOFF,ZOFF):FY=Td World
Y(O):FZ=Td World Z(O) Td Face
ALIEN,FX,FY,FZ
```

The pilot would now seem to turn away from the ship in a random direction, just like a human player.

We'll now progress to the original point of this article, and start work on creating a realistic attack manoeuvre for our aliens.

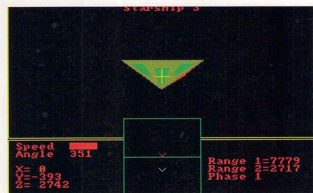
First though, we'll need to solve a rather thorny problem. The Object Modeller automatically saves all our objects just as they appear on the screen. Normally, that's fine, but sad to report, our game ship is currently facing the wrong way.

This will make our attack patterns look very silly indeed, as our aliens will now appear to be moving backwards towards the player's ship!

There are several options:

1 Load the game ship into OM and turn it so that it's pointing away from the screen. The file you'll need is GAME_SHL, and it can be found in the DEMOS/OBJECTS directory. Save it onto a fresh copy of the STOS 3D INSTALLER disk and you'll be ready to go.

2 Cheat shamelessly! Replace the game ship with an object such as ROCKET which is already facing in the correct direction!



Next month we'll slot all the pieces together to finish our game

3 Control the objects manually. As I'll be showing you next month, it's possible to override the existing direction settings, and turn our objects by hand.

We'll now simulate a simple dive bomber. The movement will be divided up into four phases.

The attacker will be initially positioned at point A, just in front of the player's ship. He'll then turn to face B, and dive directly towards us, firing continually. As he passes below our ship, he'll set course for C, and move away until he reaches a safe distance behind our player.

He'll now repeat the original attack pattern in reverse. The run will originate at C, immediately behind the ship. The alien will then duck towards our player in a sneak attack, turning to his starting position at B.

Since we'll be generating these effects using TD FACE, all movements will be relative to the player's ship. So as the player moves through space, the attacker will automatically change direction to keep in step.

This will have a massive effect on the final flight pattern, which may look very different from our simple diagrams.

We'll now have a detailed look at how these phases might be implemented within our game.

Phase=0

The attacker is moving from B to A. When he gets there, the run will commence, and he will steer to a position just below the player's ship.

```
1310 if PHSE=0 and RANGE>15000 then
YOFF=-2000 : ZOFF=0 : PHSE=1
```

The variables XOFF, YOFF, and ZOFF hold the distances from the target position to the player's spaceship, measured along the local axes X,Y,Z. RANGE contains the straight-line distance between the attacker and the ship.

It's generated automatically as part of our original scanning procedure. Note that if we initially place the ship 16,000 units ahead of the player, line 1310 will be triggered immediately at the start of our program. So the attack run will commence directly from phase 1.

Phase=1

Our enemy is now moving quickly towards the player. When he's at the

midpoint, he'll turn to a point far behind our ship.

```
1320 if (VIS=0 and RANGE<2000) and
PHSE=1 then YOFF=0 : ZOFF=-2000 :
PHSE=2
```

VIS is a variable holding the results of our previous TD VISIBLE call. It would be set from the scanner using a line such as:

```
715 VIS=Td Visible(0B)
```

Phase=2

This signals the retreat. The attacker is currently proceeding towards point C. He'll change course when he reaches a safe distance from the player.

```
1330 if PHSE=2 and RANGE>15000 then
YOFF=-2000 : ZOFF=0 : PHSE=3
```

Phase=3

Our attacker has now turned around, and is heading back towards the midpoint at C. Once he's there, he'll head towards his initial starting position.

He'll now repeat the manoeuvre from phase 0.

```
1340 if (VIS=0 and RANGE<2000) and
PHSE=3 then YOFF=0 : ZOFF=2000 :
PHSE=0
```

After lines 1300-1340 have selected the direction, we can turn the attacker like so.

```
1345 Rem turn attacker to selected
point
1350 VX=td world x(O,XOFF,YOFF,ZOFF) :
VY=td world y(O) : VZ=td world z(O)
1360 td face OB,VX,VY,VZ
1365 rem move forward
1370 td forward OB,200
```

And here's the finished routine:

```
1298 rem Simple version of swoop
1300 locate 26,22 : print "Phase": PHSE
1305 rem Move towards the player at
start
1310 if PHSE=0 and RANGE>15000 then
YOFF=-2000 : ZOFF=0 : PHSE=1
1315 rem Turn at midpoint B
1320 if (VIS=0 and RANGE<2000) and
PHSE=1 then YOFF=0 : ZOFF=-2000 :
PHSE=2
1325 rem If we are at the far end of
the dive (C) then turn towards the ship
1330 if PHSE=2 and RANGE>15000 then
YOFF=-2000 : ZOFF=0 : PHSE=3
1335 rem turn at midpoint B from behind
1340 if (VIS=0 and RANGE<2000) and
PHSE=3 then YOFF=0 : ZOFF=2000 :
PHSE=0
1345 rem turn attacker to selected
point
1350 VX=td world x(O,XOFF,YOFF,ZOFF) :
VY=td world y(O) : VZ=td world z(O)
1360 td face OB,VX,VY,VZ
1365 rem move forward
1370 td forward OB,200
1380 return
```


STOS 3D Made Easy

Arming the aliens

Sadly, this is the final instalment of the present series. All that's remaining is to add a few simple routines to control the laser, power up the shields, and arm the attackers.

But first, we'll round off the section on attack patterns from last month. As you'll recall, we exploited the TD FACE command to target our objects towards the players spaceship.

Unfortunately, TD FACE wasn't compatible with our original GAME_SHIP, because it made dangerous assumptions about the way our objects had been stored on the disk. So we were limited to boring objects such as ROCKET or DICE, which had been saved in just the right orientation.

In order to sidestep this restriction, we'll have to forget about TD FACE completely, and control the flight pattern directly from our program.

Our new routine should perform the following tasks. First, it will have to find the angles from the source object to the destination point. Then, it can then use these angles to turn the source object towards our intended target.

We'll can accomplish this procedure using a combination of the TD BEARING and TD ANGLE commands.

TD BEARING takes an object and calculates the bearings and range from the selected target area. The bearing is supplied as two angles. Angle A holds the vertical elevation, and Angle B returns the horizontal swivel.

We can think of the source object as the barrel of a gun. This can be aimed either horizontally (left to right), or vertically (up and down). You can see the idea from Figures 1a and 1b.

TD BEARING can be called up in a variety of ways. The simplest version is:

```
BA=TD BEARING A(object,target) : Rem
Gets the vertical elevation.
BB=TD BEARING B(object,target) : Rem
Gets the horizontal rotation.
BR=TD BEARING R(object,target) : Rem
Returns the range, just like TD RANGE
```

These functions provide the bearings between any two objects in our game. It's also possible to get the bearings from an specific point in the 3D universe, using a second, more advanced version, of these commands:

```
BA=TD BEARING A(object,wx,wy,wz)
BB=TD BEARING B(object,wx,wy,wz)
BR=TD BEARING R(object,wx,wy,wz)
```

In the final part of his STOS 3D tutorial, Stephen Hill shows you how to shoot and be shot at.. and gives you a shield for your safety!

wx, wy, wz now represent the world co-ordinates of the selected target.

Each TD BEARING function automatically calculates all three values in advance. So after we've chosen the initial target area, we can read out the other bearings directly:

```
BA=TD BEARING A(object,target)
BB=TD BEARING B(0)
BR=TD BEARING R(0)
```

Once we've found the direction, we can then turn our attacker using the TD ANGLE command.

Here's how this might look from our Basic program:

```
2000 BA=TD Bearing
A(MISSILE,TARGET):Rem MISSILE and
TARGET are object numbers
2010 BB=TD Bearing B(0):Rem using the
fast form of the instruction
2020 Td Angle MISSILE,BA,BB,0:Rem Turn
the missile to the new orientation
2030 Td Forward MISSILE,100
```

TD ANGLE should already be pretty familiar, as we've used extensively in our previous examples. Its action is to rotate an object through the selected angle.

Used together, TD BEARING and TD ANGLE have exactly the same effect as a single TD FACE command. So they can be substituted straight into our existing attack sequences without any fuss.

What's more, since we've taken control over the current movement direction, it's now trivial job to fix up our aiming system so that it's compatible with our original GAME_SHIP.

All we have to do is flip our objects by 180 degrees before they are displayed and return them to their original headings after the final TD REDRAW command.

We can reverse our objects with:

```
td angle OB,TA+180*182,TB,0
```

Where TA and TB hold the new heading we've chosen for our attacker.

Now that the attacker is facing in the correct direction, we can propel it

towards our target using TD FORWARD. But as it's been flipped around, we'll have to reverse the distance value like so:

```
Td Forward MISSILE,-100: rem Moves the
object BACKWARDS!
```

If we left the system as it stood, we'd get into real trouble the next time we took our bearings, as the TD BEARING command would read all our values from the newly reversed object. So it's a good idea to turn the object back to its original heading at the start of the loop.

```
Td Angle OB,TA,TB,0
```

Here's an example of this system for you to play around with. It's adapted from last month's example.

```
1371 Rem Turn OB so that it's facing
its original heading.
1372 Rem I'm afraid I missed it from
last months example!
1373 Td Angle OB,TA,TB,0
1374 TA=td bearing a(OB,VX,VY,VZ) :
TB=td bearing b(0):Rem Get the new
bearing
1375 Rem Turn the attacker so that
it's facing AWAY from the player!
1376 td angle OB,TA+32767,TB,0 :Rem
32767=180 degrees in Voodoo Rotation
units
1377 td forward OB,-200 : Rem Move the
attacker BACKWARDS!
```

This listing is slightly different from the original demo, as I'd forgotten to include the vital TD ANGLE command at line 1373. Whoops!

Well, that's the attack patterns taken care of. Let's return to our program, and quickly develop it into the finished game.

We'll begin with the shields. Each object will be surrounded by an imaginary force-field, which will provide limited protection from enemy laser fire.

After a few shots, the shields will be worn down to nothing, leaving our object completely vulnerable. So the next successful hit will destroy it utterly!

First, we'll define a few arrays to hold the required shield values:


```
56 dim
SHIELDS(NOBJECTS),MXSHIELDS(NOBJECTS),
TSHIELDS(NTYPES)
```

The SHIELDS array will hold the number of hits remaining to our shields. If it gets below zero, the object will be destroyed.

MXSHIELDS will contain the maximum shield strength of our current object. This will be used as part of the regeneration routine I'll be showing you later.

TSHIELDS will store the maximum shield strength of each TYPE of object in our game.

We'll handle the shields by redefining our original firing system:

```
530 If BEAM and TARGET then Gosub 1500
```

This calls up a brand new subroutine at 1500. Its job will be to deal with an attack, and take the appropriate action.

```
1500 SHIELDS(TARGET)=SHIELDS(TARGET)-1
1510 boom 1514 rem Check for a kill
1515 if SHIELDS(TARGET)>=0 then return
1520 inc HITS : rem increase score!
1522 rem If the object is a gravitic
mine, it will explode violently!
1523 rem see lines 1560-1570
1525 if TACTICS(TARGET,0)=PROX then
gosub 1560 else locate 26,23 : print
"Target gone!"
1530 play 40,0
1535 screen swap : wait vbl : wait 50
: rem Display messages
1536 rem DESTROY attacker, and gener-
ate a new one
1540 td kill TARGET
1541 rem Randomly choose a new ship
type and move it into position
1542 rem Ensure that there's a maximum
of one mine per wave
1543 if OBTYP(3-OB)=1 then
OTYPE=rnd(NTYPES-1)+1 else OTYPE=1
1545 DIST=22000 : REP=TARGET : gosub
1800
1550 return
```

Lines 1541-1545 replace the defeated opponent, and execute a new routine at 1800 to position it in our 3D game world.

The TD KILL command removes our target object completely from the game. But it leaves the definition in memory, ready and waiting for the next attack run. Just to be on the safe side, we'll rewrite our original scanning routine like so:

```
480 for OB=1 to NOBJECTS
481 rem Stops the scanner displaying a
DEAD object
485 if SHIELDS(OB)<0 then goto 510 :
rem NEW
486 VIS=td visible(OB) : rem NEW!
490 RANGE=td range(O,OB) : rem Get
distance between objects O and OB
500 if RANGE<=MRANGE then gosub 570
501 rem NEW! Moved from 716 so that
the movement routine is called
502 rem even if an object is out of
range.
505 gosub 1000 : rem Jump to combat
routine
510 next OB
```

We can now call up the shield regenerator I promised you earlier.

```
380 if timer>250 then gosub 1600 :
timer=0
```

TIMER is a STOS Basic function that increases by one every 1/50 of a second. The routine at 1,600 will therefore be executed at regular five-second intervals.

```
1590 rem Replenish shields
1600 for OS=0 to NOBJECTS
1610 if SHIELDS(OS)<MXSHIELDS(OS) and
SHIELDS(OS)>=0 then inc SHIELDS(OS)
1620 next OS
1630 return
```

While we're at it, we'll also provide a visible read-out of the player's shields on the control panel.

```
404 locate 26,18 : print "Shields";
405 if SHIELDS(0)>0 then ink 5 : bar
264,144 to SHIELDS(0)*5+264,150
```

We'll now add a small restriction to our laser. This will ensure that the player can't kill off an attacker by simply holding down the Fire button. In real life, lasers get very hot after use, so it's fair to assume that there's an absolute limit to the number of continuous shots by the player.

We can simulate this effect by defining some new variables:

MXHEAT will hold the maximum number of successive laser bursts.
HEAT will store the number of shots remaining to our laser.

```
37 MXHEAT=7 : rem Sets the maximum
number of continuous shots
```

We'll then include these factors into our original firing system with:

```
235 if fire and HEAT<MXHEAT then
BEAM=true:inc HEAT:wait 5 else
BEAM=false
```

And cool down the laser at regular intervals using:

```
375 inc COOL : if COOL>7 and HEAT>0
then dec HEAT : COOL=0
```

Lastly, we'll display the HEAT indicator on the screen with:

```
421 locate 26,19 : print "Laser "; :
rem NEW LASER read-out
422 if HEAT>0 then ink 5 : bar 264,152
to HEAT*7+264,158
```

The final step, is to give our aliens the ability to fire back! A shot will be fired if the following two conditions have been met.

1. There's been a reasonable delay since the last attack.
2. The player's ship is directly in front of our attacker.

The firing rate can be limited in several ways. One obvious possibility is to use

a simple RND function, such as:

```
If Rnd(100)=RATE Then....
```

However, it's more fun to adjust the firing rate depending on the range. This will require us to generate a fancy decision value such as:

```
ACC=(RANGE/5000)+7
If rnd(ACC)<=THRESHOLD Then ...
```

ACC varies between 7 and 11. It's used with the RND function to control the probability of a shot being fired.

The THRESHOLD can be anything you wish. The higher the value, the more likely the attack will be launched. In my example, I've set it to two, so the odds of an attack will vary between 3/7 and 3/11.

Note that there's absolutely nothing special about the ACC or THRESHOLD values used in this example. They were chosen purely by experimentation, and can be adjusted to produce your own individual firing patterns as required.

We'll now have a bash at checking whether the player is actually in front of our attacker. This can be tested with the TD VIEW function, which converts a series of world co-ordinates into the equivalent local versions.

We'll use it calculate the local Z co-ordinate of our ship as seen from the attacker. If it's greater than zero, we'll immediately know that the player is in under the enemies sights. So the attack can commence!

Here's the full routine::

```
1401 rem NEW fire control!
1402 rem FIRE off a laser if the
conditions are met
1403 ACC=(RANGE/5000)+7:INFRONT=td
view z(OB,SHIPX,SHIPY,SHIPZ)
1404 if rnd(ACC)<=2 and INFRONT>0 then
gosub 1700
```

Once we've fired off a shot, we'll naturally want to check whether it's hit our ship. We'll link this to the RANGE by adapting the previous firing routine like so:

```
1700 shoot
1710 ACC=(RANGE/1000)+1: rem ACC
varies between 1 and 25
1716 rem SCOL flashes the screen to a
new colour
1717 rem The chances of a hit vary
from 100% to about 12% depending on
range
1720 if rnd(ACC)<=2 then SCOL=15 :
SHIELDS(O)=SHIELDS(O)-1
1730 return
```

And that's it! You can find the completed game on next month's CoverDisk. I appreciate that it's not exactly up to Elite standards, but then again, it's only a tiny fraction of the size!

If you're prepared to add in your own attack patterns and ship types you should be able to generate some amazing games with this system. Best of luck!

**Next
month
on sale
Sept 2nd**

The CoverDisk

specials

28 Desktops

We inspect the replacements that will give your ST bright new icons and a performance boost

32 Doing the business

Not all printing bureaux use Macintoshes or PC-compatibles. Two firms say why they couldn't resist the Fuji badge

43 Buyers' guide

A complete guide to all ST software applications - their names, features and cost

reviews

44 Direct-to-disk

Why the Falcon will be so important for musicians, including a mini round-up of dtd programs

36 MultiTOS

We test Atari's multitasking system for the ST and Falcon and explain what it will do for you

40 Pad 2

A shareware vector graphics program, now with even more useful features

58 SpeedoGBOS

Atari's first scalable type handler is found to be one of the most promising ST and Falcon programs of the year

READER OFFERS

18 Buy a manual for this top programming utility and learn how to get your STOS games playing faster

92 Missed an opportunity?
Time's running out for completing your Video Master, K-Graph and other Atari ST User giveaways

regulars

7 News

Big Jaguar announcements, Atari show to be held in August, plus much more

10 Euro news

Illegal modem sales ban, 3K ComputerBild close and all the rest of this month's news from across the Channel

12 CoverDisk

A guide to getting started with the fantastic programs packed on to Britain's best ST CoverDisk

16 Disk tutorial

How to get STOS games working blindingly fast with this month's great giveaway, STOS Compiler

38 Write now!

The place where you can have a say about anything relating to Atari and their supporters

52 Public Sector

Ray-tracing, arcade action, puzzles, invoicing and education fun - all additions to this the public domain

56 Advice Service

You quiz our experts about SCSI hard drives, maths co-processors, extended format disks and more

61 C programming

Why programmers should make their programs easy to use and how they should go about it

82 Subscriptions

Stay up to date with 12 issues of Atari ST User and pick a free gift for your computer

84 Classifieds

Your first stop, whether you want to sell old equipment or pick up an ST bargain

86 Aspects

Six pages devoted to the most popular ST applications - music, emulation, comms, DTP and business

88 Competition

Three copies of top music program Slave Driver must be won in this great £450 giveaway

CHAIRMAN Derek Macklin
MANAGING DIRECTOR Brian Bloomfield

Member of the Audit Bureau of Circulation

Atari ST User is an independent publication and Atari are not responsible for any of the articles in this issue or for any of the opinions expressed.

© 1993 Europress Enterprise Ltd.
No material may be reproduced in whole or in part without written permission.
While every care is taken, the publishers cannot be held legally responsible for any errors in articles, listings or advertisements.

EUROPRESS
ENTERPRISES

Printed in the UK by Carlisle Web Offset, Carlisle, Cumbria



**Design
your
own games using STOS
and then make them
much faster with STOS
Compiler - two complete
programs given away
free on this month's
CoverDisk
PLUS:**

● Increase your chances of a pools win with the latest version of Bay Computer's Pools Predictor and a complete list of this season's football fixtures from Dr Fox



The DISK starts on page 12

disk COVERDISK

This month we have a double STOS giveaway with the complete versions of STOS and STOS Compiler Interpreter that will allow you to create stunning arcade games, demos and even serious applications

STOS Compiler

The complete and unrestricted STOS compiler that transforms your STOS Basic programs into pure, blindingly fast machine code

Programmed by: Francois Lionet
Configuration: Requires STOS Interpreter
Filename: X_COMPILER.TOS -
Self-extracting archive

There can't be many ST users who have not heard of STOS. Being a Basic-like programming language, it provides the user with all the commands required to enable quality arcade games to be written without the need for experience in the intricacies of assembly language.

However, STOS is an interpreted language and therefore suffers from a certain lack of speed in performance. This is because each command in your program has to be converted into code which the computer understands every time it is encountered.

A good analogy is to think of your basic program as a page of text written in French and being read by a person with no knowledge of the French language whatsoever. The person can interpret each word as it is encountered, using a French dictionary, or, the entire document can be translated into English by someone who can speak French.

Of course the latter method is more efficient and preferable as, once translated into English, it can be immediately read without

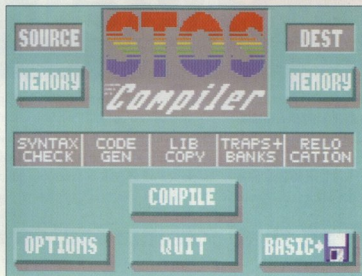
any intermediate interpretations whatsoever. Essentially, the STOS Compiler is the translator, turning the basic program into pure machine code ready to run. This results in a vast increase in speed due to the omission of the interpretation process.

Once installed, the compiler is accessible from within the STOS editor and is also extremely easy to use with the main features of the compiler being controlled through a collection of five buttons. It is possible to compile programs to or from memory or disk and an options menu allows you to configure the way the compiler behaves.

For a quick guide to getting the compiler up and running, turn to page 16 which details the installation procedure and runs through an example compilation procedure.

Single-sided service

Single-sided drive users wanting the contents on this month's disk on single-sided disks should send their original CoverDisk and a cheque for £1.50 - payable to Atari ST User - to SSD Service, Atari ST User, Europa House, Adlington Park, Macclesfield SK10 4NP.



Turn your STOS basic games into very fast machine code using the STOS Compiler.

STOS - The Games Creator

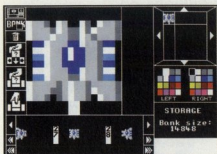
Programmed by: Francois Lianet
& Constantin Sotiropoulos
Configuration: All STs (except STOS
v2.05/6) - all resolutions
Filename: X_STOS1.TOS, X_STOS2.TOS,
X_STOS3.TOS - Self-extracting archives

Together with the compiler, the STOS interpreter will allow you to easily create some pretty fast games and programs to compete with many commercial titles on the market.

Just take a look through much of the PD software available and you'll find that a large proportion is written in STOS, proving just how powerful and popular this programming language actually is.

Although there is a comprehensive list of commands (over 340) that make creating games easy, such as those for manipulating

Yes, that's right, we are giving you the fully working and latest version of the STOS interpreter to complement the compiler! Aren't we generous to a fault?



At the press of a couple of keys, you can bring up the sprite editor to create the sprites for your game

sprites, graphic screens, animation and much more. STOS also provides commands which make creating drop down menus, file selectors and windows extremely easy, enabling more serious applications such as wordprocessors or spreadsheets to be developed.

The three archived STOS files on our CoverDisk need to be copied to their own blank disk. X_STOS1.TOS contains the main STOS interpreter files and is auto-booting (label this disk STOS Interpreter), X_STOS2.TOS contains STOS accessories which can be loaded along with STOS and which provide useful features (label this disk STOS Accessories).

The last disk contains example games written with STOS (label this disk STOS Games).

The accessories make program development much easier and include such utilities as a sprite definer, map editor, music editor, character editor, icon editor and others (including the compiler). These accessories are available from within the STOS editor (subject to available memory) and are available by pressing the Help key.

Of particular interest to game designers are the sprite and map editors. These two utilities allow you to create, edit and save sprites and graphic blocks. Sprites are generally the things you see moving on the screen, such as a spaceship controlled by the player and even the bullets which are fired.

The map editor allows you to create "building blocks" which you can create a scrolling terrain or backdrop to your game which the sprites move over.

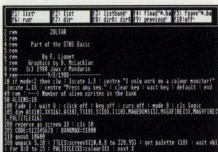
Because these utilities

are available from within the STOS editor, you can be writing a piece of code that controls a sprites movement, then switch to the sprite editor to actually draw the sprite which the code will manipulate, then, after saving the sprite, flick back to the editor to finish of the code or test it.

This level of integration means that development of a game is both quick and efficient and is a major reason behind the success of the STOS language.

Once you have copied each of the three files to a blank formatted disk, execute each one in order to extract them. As each archived program finishes extracting, be sure to delete the archived file from the blank disk as it is now obsolete and is just wasting space.

With all files successfully extracted, turn to page 16 for details on how to install the STOS compiler into the STOS interpreter



The STOS Basic editor is where you enter your program code for the next blockbuster game

STOS Basic sprite commands

Take a look at the comprehensive range of sprite commands available which make manipulating sprites a real breeze

SPRITE	Draw a sprite
MOVE	Start/stop movements
MOVE X	Move sprites left and right using interrupts
MOVE Y	Move sprites up and down using interrupts
ANIM	Animate a sprite
PUT SPRITE	Copy a sprite to the screen
GET SPRITE	Make a rectangular section of the screen into a sprite
UPDATE	Update sprites
AUTOBACK	Switch off link between sprite background and real screen
X SPRITE	Get X co-ordinate of sprite
Y SPRITE	Get Y co-ordinate of sprite
MOVON	Check if sprite is currently in motion
COLLIDE	Test for sprite collisions
LIMIT SPRITE	Limit sprite movements to a part of the screen
ZONE	Test if sprite enters a specified rectangular section of the screen
SET ZONE	Define up to 128 rectangular zones
RESET ZONE	Clear currently specified zones
REDRAW	Redraw a sprite
PRIORITY	Change sprite priority
DETECT	Detect pixel under sprite
SYNCHRO	Synchronise sprite with scrolling background



Using the map editor, you can create detailed backdrops for a platform game or a vast terrain for a role playing game

Problems with your CoverDisk?

With thousands of disks being duplicated each month, a few will inevitably be faulty. Fortunately, we can help you. Send the faulty disk to:

PC Wise, Dowlais Top Park, Dowlais,
Merthyr CF48 2YY

A replacement will be sent free of charge but please allow 28 days for delivery.

CoverDisk Hotline

Hotline number: 0625 859766, 2pm to 5pm
every Wednesday.

If you have any other problems with the CoverDisk, our technical editor, Darren Evans, will be available on the Atari ST User CoverDisk Hotline. Help is only available within the times specified, and no advice will be available outside these hours.

Pools Predictor v4.11

**Odds against you winning millions on the pools?
Then lower them and improve your chances with
this excellent pools prediction program**

Programmed by: Brian Campbell
of Bay Computers
Configuration: All STs - medium
or high resolution
Filename: X.POOLS.TOS -
Self-extracting archive

We all dream of winning the jackpot on the pools, but the plain fact is that the odds of actually winning the damn thing are stacked against us.

Although using this program will not guarantee a win, it will certainly lower the odds a little and improve your chances of success.

Some of you may remember that this great program appeared on a previous Atari ST User CoverDisk (October '92) in demo form. The author was so impressed with the response from readers wanting to register that he has decided to allow the complete version to be given exclusively to our readers.

The Pools program is licensed shareware which means it is only available from authorised distributors, and to receive the manual and annual updates, you can register for a mere £15.

Many great programs are available as shareware and to ensure continued release and development of such programs, you should send the registration fee. This also ensures you get updates of the program.

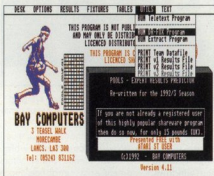
This new version has been improved and is much simpler to use than earlier versions. Added features include the ability to configure the program to accommodate almost any league in the world with little effort.

It can now accommodate the additional Scottish division commencing 1994/5; any results entered can be transferred to the Dr Fox Football Annual program (if present); an additional 390 teams have been added to the team database; the results database now contains over 35,000 results; and an automatic success rate and monitor facility on all predictions is available.

Also included is a demo of the Dr Fox Football Annual, a powerful statistical analysis program which can be used to analyse teams,

FOOTBALL EXCLUSIVE!

Included with the Pools Predictor and the Dr Fox program are the up-to-the-minute league fixtures for this season, so you won't have to enter the information yourself.



The ability to execute the Dr Fox Football Annual from within Pools Predictor makes for an extremely powerful combination

Pools memory requirements

Although Pools can be used with a 512k ST, 1Mb is recommended. The following functions are unavailable from within the program on machines with only 512k. However, they can be run as standalone programs:

- Reset League Tables
- Run Teletext Program
- Run Dr Fox Annual
- Run Extract program

fixtures and divisions with the option of creating graphs and printing the information.

What makes the Pools Predictor and Dr Fox's annual even more powerful is the fact that they can be used together as an integrated package (already set up for you on the CoverDisk) with the ability to run the Dr Fox program from within the Pools program (via the *Utils* menu) and the ability for data to be transferred between them.

Also, the author of the Dr Fox utility has a licence to publish the official fixtures of the major divisions, meaning users have the latest and up-to-date league data.

Leagues covered include the FA Premier League, Football League, Scottish Football League, GM Vauxhall League, Northern Premier League Premier Division, Diadora League Premier Division and the Beazer Homes League Premier Division.

The full version of the Dr Fox Football Annual is available for £7.99 and comes with all last season's results, all this season's fixtures and all results up to the date of dispatch. For further details on registering for both the Pools program and the Dr Fox program, refer to POOLS.TXT and READ_ME files respectively.



Using the CoverDisk

Write-protect your CoverDisk now by sliding the write-protect tab so you can see through the hole at the top right corner of the CoverDisk.

In order to fill the disk with as many programs as possible, ST User employs disk compression techniques when placing large programs on the CoverDisk.

These files are termed "archived files" and cannot be directly executed from the CoverDisk. Archived files are identified by their names beginning with *X_*.

Any files that do not begin with *X_* are actually folders containing files which are not compressed.

These uncompressed files should also be copied to a blank disk. Following are a series of step-by-step instructions for using the CoverDisk.

Important: Make a backup of your original CoverDisk and put it in a safe place.

Using archived files

Note: When copying files or disks, the ST refers to disk A and disk B in its on-screen instructions. Disk A refers to the disk being copied while disk B refers to the disk being copied to (usually a pre-formatted blank disk).

1. Format a disk.
2. Copy the required archived file to the formatted disk.
3. Run the copied archived file. It will now automatically "extract" to its full size.
4. Once complete, delete the *X_* filename as this is no longer required. The formatted disk should now contain all the files that were in the archive file.

Normal uncompressed files

1. Format a blank disk.
2. Open the required folder and copy all files in the folder to the formatted disk.
3. You can now immediately run the program's as is.

Highlighting and dragging files and icons

1. Click once on the item, it should turn black (the item is now highlighted).
2. If you keep the left mouse button held down on the item, a dotted outline of the item appears and you may then reposition it by moving the mouse to the required position (effectively dragging the icon) and releasing the mouse button.

If you are new to the ST and did not understand some of the terminology in the preceding steps, there now follow an explanation of some basic techniques needed.

Formatting a disk

1. Place the disk to be formatted in the drive.
2. Highlight the drive A icon.
3. Move mouse pointer up to the FILE menu at the top of the screen and select FORMAT.
4. A box will appear. Click on OK.
5. Another box will appear. Ensure DOUBLE SIDED is selected then click on OK. The disk is now being formatted.

Copying files to another disk

1. Select the file to be copied by highlighting it and dragging it to the disk it is to be copied to.
2. A box entitled COPY FILE(s) appears. Click on OK and follow instructions on the screen exchanging disks as requested.

Once you have extracted the three STOS archived files, you should have three disks labelled STOS Interpreter, STOS Accessories and STOS Games. If you haven't done this yet, turn to page 12 for instructions.

You must now copy the file X_COMPILER.TOS from the CoverDisk onto a blank disk labelled STOS Compiler. After you have done this, run the file X_COMPILER.TOS and the compiler will be extracted.

After the extraction has finished, delete the file X_COMPILER.TOS from the STOS Compiler disk. You are now ready to install the compiler onto your STOS Interpreter disk.

Turn off your ST for approximately 30 seconds (a good habit to adopt to avoid viruses), insert the STOS Interpreter disk and turn the ST back on.

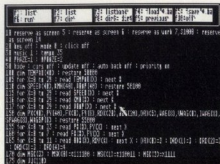
After a short time, you will see the STOS loading screen. Eventually, you will be presented with the STOS editor and the word OK, along with a flashing cursor, will appear indicating that STOS is ready for your commands.

COMMAND KEYS

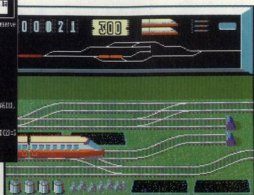
This is where all the action takes place. You can enter your program instructions, load, save or edit basic programs and also call up the various accessories available, such as the sprite designer, map editor and, after we have finished here, the compiler too.

Remove the Interpreter disk and insert the Compiler disk into the drive. At the top of the screen you can see a selection commonly used commands with a function key number next to them. To load a STOS basic program, press the F4 key which brings up a box for selecting the file you wish to load.

Click on the file STOSV206.BAS and then Return. This loads the file into memory. To verify this, type the word LIST and press Return. You will see that lots of commands will scroll past (press Space to pause the listing if you wish). This is the program

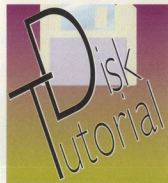


This is the STOS editor where you enter all the program code for your blockbuster game



Access resident accessories such as the sprite editor and compiler by pressing the Help key

Speedy Basic



A brief guide to using this month's double STOS giveaway, showing you how to inject some speed into your STOS Basic programs

which you have just loaded.

The actual program installs the compiler extension on to your STOS Interpreter disk and also updates your STOS version to v2.6. Don't worry if you already have v2.6 – nothing adverse will happen.

Type the command Run and press Return. The program will execute and you will be asked to insert your STOS Interpreter language disk into the drive.

Other accessories available include this powerful sprite editor. Such utilities greatly simplify game development

Do this now, then press the G key. Your language disk will now be updated and the STOS Compiler extension installed.

After the process finishes, press the Space key to end the program and you will be taken back to the editor. Press the Undo key twice to clear the screen and type the word New and press Return to clear the program in memory.

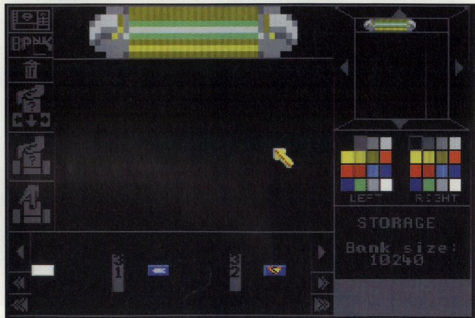
Your STOS Interpreter language disk has now been updated (if it was a early version) and also has the Compiler extension installed.

Accessing the Compiler from the STOS editor is achieved through the Compiler accessory. This, along with some others, can be found on the STOS Compiler disk and is called COMPILER.ACB. You should now go back to the ST's Desktop and copy the COMPILER.ACB accessory to the STOS Interpreter disk. Once done, load up STOS Interpreter again.

This time when the editor screen appears, you should see that the list of extensions installed includes the Compiler extension too.

There are two ways to load an accessory. One is to have an accessory automatically load when STOS Interpreter is run, the other is to load a accessory from the editor manually. The latter is achieved through the Aload command.

Insert the STOS Accessories disk and type DIR "A:*.*ACB" and press Return. This will list all accessories (which by the way are identified by the last three characters ACB) on the disk. To load a



specific accessory, for instance the SPRITE.ACB accessory, simply type Accload "SPRITE.ACB". If you then press the HELP key, a box will appear and at the bottom of the box is a list of accessories currently loaded.

As you can see, the number one slot has the sprite accessory loaded. To execute the accessory, simply press the appropriate function key, in this case, F1.

To have the STOS Interpreter load up frequently used accessories at startup, you must run the configuration program. This program is, spookily enough, called CONFIG.BAS and is found on the STOS Interpreter disk. Load this now by typing Load "CONFIG.BAS" then typing Run and pressing Return.

Shortly you will be seen various parameters, such as default resolution and default language, which can be changed to your needs. For now, do not change anything, but simply click on the Next Page box at the bottom right of the screen.

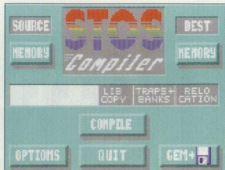
AUTO LOADING

The next screen allows you to edit the function activated by the function keys, and at the bottom of the screen, a list of loaded accessories can be seen. At the moment, this probably only contains the sprite accessory at slot number one.

To tell the editor to load an accessory automatically at startup, simply click on the blank space to the right of the required function number of your choice and then type in the filename of the accessory (including the ACB extender) which is to be loaded and then press Return. Try this now by entering COMPILER.ACB into slot number two. To save any changes made, be sure to click on the save On Disk button, otherwise all changes will be lost when you reboot.

If you now reload the STOS Interpreter, when you are presented with the editor screen, you will see that the editor will automatically load the accessories you specified in the configuration program.

We will now run through a compilation process to show just how simple it is to use the STOS Compiler. What we will do is use one of the example games on the STOS Games disk and compile it into pure



Once installed, the STOS Compiler is extremely easy to use and allows compilation to either memory or disk



STOS can be configured to your own personal preferences with the ability to automatically load frequently used accessories

Hard drive owners

For those of you lucky enough to have a hard drive, you can copy the contents of the STOS Interpreter, Compiler and Accessories onto a partition so that disk swapping is avoided.

machine code, but first of all, we will see how fast the game is before to compiling it. Insert the STOS Games disk and press the F4 key. A file selector box will appear showing three folders called Bullet, Orbit and Zoltar. Click on the Bullet folder and select BULLET.BAS, then press Return. This will load the Bullet basic program into memory.

When it has loaded, type Run to execute the game. Those of you with only 512k may have to clear any accessories which may be loaded to avoid any out of memory messages. To clear accessories, type ACCNEW before you run the game.

Play the game for a while and keep in mind the speed of animation. Now we will compile it and see how it compares. Press Control and the C key at the same time to exit the game and then press the Undo key twice to clear the screen.

Put your STOS Interpreter disk in the drive and load the compiler accessory by typing Accload "COMPILER.ACB". Now press Help and press the appropriate function key to execute the compiler.



You will probably be asked to insert the STOS Compiler disk at this point. Do so and eventually the compiler menu screen will appear. As you can see, there are a mere six buttons to cope with. Under words Source and Destination you can see two buttons marked Memory. These specify where the program to be compiled is (Source) and where it is to be compiled to (Destination).

Click on the button under the word Destination and it will change to Disk. This means we are going to compile the basic program and save it on a disk. Next, click on the button marked Basic and it will change to Gem. This changes the type of compiled file produced. A Basic compiled file needs to be loaded into STOS Interpreter and executed from there, whereas a GEM-type compiled program is a standalone executable file.

Now click on Compile and you will be asked to specify a filename for the compiled program, type in BULLET.PRG and click on Return. The compilation process will now begin with a white status bar indicating the progress.

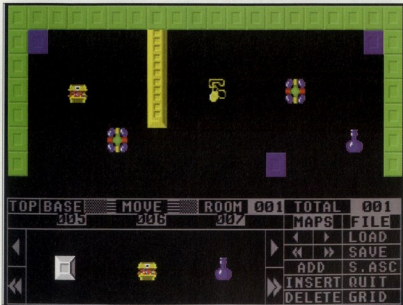
FINISHING OFF

After the compilation process is finished, the compiler prints the statistics regarding the size of the compiled file and the amount of time taken to compile it. Click the mouse button then click on Quit to exit.

Now type System to inform STOS Interpreter that you want to return to the ST's desktop and press the Y key to confirm. If you examine the disk in the drive, you will see that the file BULLET.PRG is present. This is the compiled version of the BULLET.BAS program file. You should now copy the BULLET.PRG file into the BULLET folder.

To run the program, double-click on the BULLET.PRG file. As you can see, there is a marked increase in speed over the uncompiled version and the whole compilation process is extremely easy.

To get the most out of the compiler it is best to use the manual which contains detailed information such as optimising the compiler to take advantage of RAM disks, a tutorial section, information on the compiler options available and much more. For information on how to obtain this manual, turn to page 18.



This map editor accessory lets you create backdrop graphics for platform games and even vast landscapes for an RPG