STOS Newsletter

Issue 9 • STOS User Club

◻ Editorial

Welcome to another STOS Club Newsletter ! This issue we have all the regular articles for the absolute beginner to the expert machine coder. The now regular contribution by the famous Doctor Speck O.D.D and plenty of information on new STOS products. If you have only just got STOS, or have only started your subscription to the newsletter this year, you might want to know how to obtain issues 1-6. Simple, just send a cheque or postal order for £6 to Sandra Sharkey at the P.D Library address.

◻ AMOS

By the time you read this, AMOS (The Amiga version of STOS) should be about to be released, along with the AMOS club. Keep your eyes open for some special collaboration between the two clubs, especially if you know someone with an Amiga, as there will be articles on transfering data between the two machines, and most of the software developed on one machine will be written for the other.

You will have noticed that this issue is almost a month late, this has been done to give us time to write issue 1 of the AMOS newsletter, and to make it so that the two newsletters are produced on alternate months.

◻ The Games Maker's Manual

The response to our Games Maker's Manual Book+Disk offer was amazing, within days of the newsletters going out, I had to re-order more books (Thus causing some delays). By the time you get this newsletter, all the current orders should be covered. The offer still stands, £11.95 for both the book and listings disk (or £2 for the disk if you already have the book), but the 48 hour turnaround (Put in by a somewhat optimistic Chris Payne !) has been adjusted to around 2-3 Weeks.

◻ TOME V2.1

The upgrade to the TOME system is expected to be available in mid to late July. V2.1 includes an improved (50% more features) editor, more efficient extension and extra commands.

☞Features are:

Editor has full basic artwork facilities, including Cut/Paste (Pixel accurate) and Tile Flip/Rotate, and can work on 4 maps at once with instant autolocation. Includes grid overlays so you can see exactly what you are doing. Map scrolling routines are just over 2 times faster than V2.0 and also include single pixel smooth scrolling (Fully documented) ! The new version includes 2 new demo games (Both compiled and source

Inside this issue:

NEWS

code) which feature high speed single pixel scrolling in all directions.

The retail price of TOME V2.1 is now £19.95, but the package is still £14.95 to STOS Club members, and if you already have V2.0 then you can upgrade quite easily.

1) Make backups of both of your TOME disks (You should have done this anyway !)

2) Send both MASTER disks and a cheque/P.O for £1 (to cover the cost of the new manual and postage) to me at the usual address.

3) Sit back and relax (You can use your backup disks until V2.1 arrives)

Obviously your disks cannot be upgraded until V2.1 is finished, so it might be wise to check first to make sure the upgrade is available.

☐ The Oh Yes I Forgot To Mention Department

First a bug in the Absolute Beginers article in issue 7. (Typo error not me!) Line 120 should read:

120 ink 0: for A = 0 to 198 step 2

It would seem that the equals sign got missed out in the transfer over to the Macintosh. Hopefully most of you managed to spot it and fix it, but hopefully this will help those of you who were worried that it was your typing that was at fault !

Also connected with issue 7, the STOS WORD package has an extra hidden feature. This was not documented as it is unfinished in the freeble version, but if you press the right mouse button a scroller box appears, which lets you move up and down the whole document. When you have finished click on the O.K button (Not easy as the zone is slightly offset). Due to the fact STOS WORD has difficulty in working with the Star LC-10 printers, and because I've just got one and found out that it's got all sorts of wonderfull features, I'm converting STOS WORD to a special LC-10 version so you can use Letter Quality fonts, and so the graphics work properly. This version will be available as shareware, more news in the next issue.

☐ P.D News

Soon to be available in the P.D/Shareware library will be demo versions of Skystrike Plus and Yomo 2389 (Both 1 level playable versions). All our future games will have a 1 level playable version released on Shareware so that you can try them out before buying the actual game. These will be called (Quite logically) "SINGLES", and hopefully other games writers and software companies will use this idea also. As this editorial is always written before the final P.D/Shareware list is put together they might already be in, so check the lists !

☐ STOP PRESS !!!!!

Sky Strike Plus is about to be released by Atlantis Software at £4.99!! This will be a 1 disk version, so those of you who bought the 2 disk version through the club for £9.95 have still got a good deal. The 2 disk version has now been discontinued (all current orders have been covered though).

Page 6 New Atari User

Page 6 magazine (also known as New Atari User) regularly run a STOS feature in their ST section. Pete Hickman, the journo hack who writes the column has reviewed STOS products, P.D and just about always has some sort of program listing in his column. Well worth a read ! (This is a definite plug for Page 6, as Pete always mentions the STOS Club in his articles !)

Important Notice !

Is your copy of STOS the latest version, are you STE compatible ? Get the V2.5 update from the P.D Library !

Utilizing

Welcome to the first in a series of articles on using the various utility programs you can get for STOS. I am basing the series upon typical helpline requests, and so far most seem to have been on how to use the Total Map Editor system in various situations. So here it is, along with some memory saving tips for STOS Compiler users!

☐ The "Gauntlet" style game with TOME

In this style of game, the main character stays in the centre of the screen, and the map (and enemy sprites) moves around it. This is quite simple to reproduce and the following short program will demonstrate it. You will need a sprite for the main character (Sprite number in the variable SPR) and a TOME map & Tile screen (obviously). If all else fails you can use the map and screen from the demo game.

```
10 mode 0:key off:curs off:hide on
20 reserve as work 6,10004 : bload "MYMAP.MAP",start(6) : rem change bank length and map name to those of your map.
30 load "MYTILES.MBK",5 : rem load in a tile screen in .MBK format
40 S5=start(5):S6=start(6) : rem get bank addresses
50 map banks S6,S6,S6,S6,S5 : rem set up banks
60 map view 0,0,256,176 : rem set up view window
70 SPR=1 : X=mapx(0)/2:Y=mapy(0)/2 : rem set sprite number and X,Y co-ordinates on map
100 do map back,X,Y : screen copy back to logic :rem display map to background screen then copy it to the logical screen
110 sprite 1,128,88,SPR : update
120 If jup and y>0 then dec y
130 If jdown and y<90 then inc y
140 If jleft and x>0 then dec x
150 If jright and x<86 then inc x
160 If inkey$<>"" then end
199 goto 100
```

Upon running this program, you should have a sprite in the centre of the screen, with the map scrolling around it controlled by the joystick. This routine usually runs a little fast, so you might want to throw a few **WAIT VBL**'s in on line 115. To check what tile your sprite is standing on, you simply use the **MAP TILE(x,y)** function to return the tile number at the co-ordinate of your sprite. **MAP TILE** returns the tile value at co-ordinates X,Y however. So if you simply put the following line into your program:

```
170 T=map tile(X,Y)
```

then T would return the tile number of the tile at the top left of the screen. So you need to add an offset, as your sprite is offset into the screen. For our sprite co-ordinates of 128,88 we would need an X offset of 8 (each tile is 16x16 in low rez, so 128 pixels is 8 tiles) and a Y offset of 5 (Round down). Thus our line would be:

```
170 T=map tile(X+8,Y+5)
```

by putting the next line in, we can display the tile number in the top right of the screen

```
180 locate 30,0:Print T;" ";
```

If you want to use the powerfull Tile valuing system in TOME, which allows you to assign up to two seperate values for each tile, thus allowing you to make several similar tiles return the same value. You simply have to modify line 170 to

```
170 T=tile type(0,X+8,Y+5)
```

the tile type function is new to V2.1 of TOME, so if you are using V2.0 and haven't sent off for your free upgrade yet, use the following line:

```
170 T=peek(start(5)+32200+map tile(X+8,Y+5))
```

You can see why I put the Tile Type function in can't you?! Say, for instance you wanted to check for collision with all the walls in your map, but you have several

ARTICLE

different tiles for your walls. All you would need to do is remember a single value as "WALL" say 10. Then using the **VAL** function on the editor to assign this value to all your wall tiles. Then in the program, T will return 10 whenever you stand on a wall (at which point you use a sampled "ouch!" and return X,Y to their previous values). Try adding the following lines to try this out :

```
180 IF T=10 THEN X=XO:Y=YO :rem stick your sampled "ouch!" in here
190 XO=X:YO=Y
```

More on TOME next issue, send in your requests!

□ Compiler Tricks

If your compiled programs keep running out of memory on a 520ST try the following tricks to conserve memory.

1) Save the .PRG program in an AUTO folder, thus ignoring GEM and saving 32K!

Using the Options menu in the compiler:

2) If your program isn't using the mouse, switch off the mouse cursor on start up, and deselect the loaded mice (page 2)

3) If your program isn't using windows, then reduce the window buffer size to 8K (saving 24K). If it does use windows then reduce it to 8K+4K for every window used.

4) If your program doesn't use all the fonts or resolutions, then deselect the non-used fonts. (saves about 2K for each font)

Write in!

Please send articles and questions to:
Aaron Fothergill, N.Devon

Questions will be answered by letter if you send a stamped self addressed envelope, otherwise they may be included in the newsletter.

Asteroid Adventure

by J.A. Ure Review by Aaron Fothergill

This game is a text with a few graphicy bits adventure set on a research asteroid deep in space. You receive a distress signal in your trader ship and go to investigate. I'm not saying any more as it would give away the plot.

I am a fan of text adventures (especially Infocom stuff) as the writers tend to put more detail into the plot rather than wasting time on graphics. The plot for Asteroid Adventure is quite good, and is reasonably simple to get into, which I was pleased at, because I'm not normally brilliant at playing adventure games. Full marks also for sending me all the hint sheets, so that I got a chance to go through the whole game rather than reviewing it based on the intro. The game adds bits of graphics from time,to time. Such as when you examine an item, a small picture will appear at the bottom of the screen. Your hand held computer displays here to. On the whole the graphics aren't very good, but more importantly they add to the game rather than just being there to look pretty.

There are some good puzzles in this adventure, although a couple of them are at first a little obscure. I was pleased at actually being able to kick something and get a usefull response though, this not being normal in adventure games.

The game has its faults, the parser especially. It is one of the more simplistic types and lets the game down a bit. A text only adventure needs a good parser!

Overall, Asteroid Adventure is good fun and (apart from the parser) would be a good introductory adventure. 70%, 80% if the parser was improved!

Asteroid Adventure comes in two versions, Low rez (reviewed) and High Rez. Both are available from **J.A Ure, Birmingham**

□ Bomber 10-Liner Game

This game is the familiar one where you have to blow up buildings to clear a runway for your aircraft. Well remembered from the days of the ZX81, this game can be done in 10 lines of STOS ! All you need to do is type in the listing below and create a sprite bank with 4 sprites.

Sprites 1 & 2 the aircraft (approx 32x16 pixels size, with the hot spot on the nose of the aircraft) This aircraft should be travelling right. The game will animate between sprites 1&2

Sprite 3 is the bomb (approx 16x8 pixels size, the hotspot at the tip of the bomb)

Sprite 4 is a building section, This must be 16x16 pixels with the hotspot 1 pixel to the right of the top left.

To play the game, use the fire button to drop a bomb. If it hits a building, it will reduce the building and score 10 points. If it hits the runway, or if the plane crashes into the building, you will lose the game.

```
10 dim H(20) : mode 0 : key off : curs off : flash off: hide on:A=hunt(start(1) to start(1)+length(1),"PALT")+4 : for B=0 to 15 : colour B,deek(A+B*2) : next B : LVL=1
20 ink 0 : bar 0,0 to 319,184 : ink 1 : bar 0,184 to 319,199 : BX=999 : X=0 : Y=16 : T=0 : for A=0 to 19 :H(A)=rnd(4+LVL) : for B=0 to H(A) : sprite1,A*16+1,168-B*16,4 : update : put sprite 1 : next B : next A
30 sprite 1,X,Y,1+T : sprite 2,BX,BY,3 : update : T=1-T : if BX<320 then if 11-H(BX/16)<BY/16 then dec H(BX/16) : ink 0 : bar (BX/16)*16,152-H(BX/16)*16 to (BX/16)*16+15,167-H(BX/16)*16 : BX=999 : boom : SCRE=SCRE+10 : locate 0,0 : print "Score ";SCRE
40 X=X+LVL : if X>330 then Y=Y+min(16,LVL*4) : X=0 : if Y>=184 then 90
50 if Y>168-H(X/16)*16 then boom : goto 100
60 if fire and X<320 and BX=999 then BX=(X/16)*16 : BY=Y
70 if BX<>999 then BY=BY+4 : if BY>199 then boom : BX=999 : goto 100 80 goto 30
90 inc LVL : SCRE=SCRE+500 : goto 20
100 locate 0,10 : paper 0 : pen 1 : centre "Game Over" : wait key : end
```

□10-Things you probably didn't know about the editor of this magazine.

- Aaron lives in the middle of no-where in North Devon. He was born in Keighley, Yorkshire on the 3rd March 1967. He has only been in North Devon for 4 years since moving back to England from Hong Kong.
- He is the only person to have ever sold an origional piece of software in the notorious Golden Shopping Arcade in Sham-Shui-Po Hong Kong, where 150 shops sell pirated software under the control of the Triad gangs.
- Software written to date includes Jitterbugs (on ST Amiga Format disk 9), Skystrike, Skystrike Plus, YOMO, TOME and a little known word processor called STOS Word !
- Aaron Used to write Synthesiser editors. He is a profesional musician, with his own MIDI studio.
- He also writes for AMOS now, he wrote the Game Magic Forest and the Sprite and Map editors for the AMOS package.
- His spelling is terrible.
- Aaron's first computer was an Apple][+ clone called a Banana !
- He spends his Sundays songwriting at the beach in the summer. As that's the only time off he gets from the 2 clubs and Shadow Software !
- Aaron is a great fan of Mike Oldfield, and Steve Val. His favourite games are Olds, Dungeon Master and Super Sprint.

ARTICLE

DEGAS ELITE COMPRESSED FILE FORMAT

• By Richard Gale

Richard has sent in this listing and information on the Degas Compressed file format. In the next issue his decompressor for the TINY PIC format.

□ File Format

1 word Resolution (with high order bit set) hex 8000 = low resolution. hex 8001 = medium resolution. hex 8002 = high resolution.
16 words Colour palette, 1 word for each colour.
32000 bytes Control and data bytes (32000 bytes or less)
4 words Left colour animation limit table (starting colour numbers for each channel)
4 words Right colour animation limit table (ending colour numbers for each channel)
4 words Animation channel direction flag (0 = left, 1 = off, 2 = right)
4 words Animation channel delay in 1/60's of a second (values used 0 to 128)

□ PROGRAM LISTING

```
100 key off : curs off : flash off : hide on
110 reserve as work 10,32256 : bload "picture.pc1",10
120 SCR=physic : STRT=start(10)
130 gosub 1010
140 end
1000 rem ~~~~~~~~~~~~~~~~
DECOMPRESS DEGAS PC? FILE
~~~~~~~~~~~~~~~~
1010 REZ=deek(STRT): bclr 15,REZ
1020 if REZ<0 or REZ>2 then print "Incorrect DEGAS file" : end
1030 if (REZ=0 or REZ=1) and mode<>2 then mode REZ
1040 if mode<>REZ then print "Incorrect screen mode." : end
1050 NBPLAN=2^(3-mode) : OFSET=(2^mode)*20
1060 for I=0 to 15 : colour I,deek(STRT+I*2+2) : next I
1070 MEM=STRT+34 : SCANLINE=0
1080 repeat
1090 BUFFER=STRT+32096 : BYTE=0
1100 repeat
1110 DAT=peek(MEM) : inc MEM
1120 if DAT<128 then inc DAT : FLAG=false
1130 if DAT>128 then DAT=257-DAT : FLAG=true
1140 for I=1 to DAT
1150 poke BUFFER+BYTE,peek(MEM)
1160 inc BYTE
1170 if FLAG=false then inc MEM
1180 next I
1190 if FLAG=true then inc MEM
1200 until BYTE=160
1210 for BYTE=1 to OFSET
1220 for PLANE=0 to 3-REZ
1230 doke SCR+PLANE*2,deek(BUFFER+PLANE*OFSET*2)
1240 next PLANE
1250 SCR=SCR+NBPLAN
BUFFER=BUFFER+2
1260 next BYTE
1270 inc SCANLINE
1280 until SCANLINE=200
1290 return
```

□ Comments:

100 This removes the functions keys from the top of the screen, switches off the cursor and any flashing effects, and removes the mouse pointer.
110 Reserve an area in memory to contain the Compressed PC? file, and load the file into memory. The BLOAD"",addr command is used as the extension PC1, PC2 or PC3 is not recognised by STOS.
120 The variable SCR is set to hold the destination address of the uncompressed picture, in this case the PHYSICal screen. And variable STRT is set to hold the

position in memory of the compressed picture, defined as START(10).
130 Calls the decompress routine.
140 Ends the program.

DECOMPRESS DEGAS PC? ROUTINE

1000 Remark. Sub-routine identifier.
1010 The variable REZ is set to hold the first byte of the compressed picture, which stores the resolution. And clears bit 15 to remove unwanted data.
1020 Checks to make sure that the resolution is correct. If REZ is neither 0,1 or 2 then the file is invalid.
1030 If the current screen in either low or medium resolution and REZ is 0 or 1 then the correct MODE is selected.
1040 If the current MODE is not equal to REZ then the compressed screen cannot be shown. This can happen if trying to view a monochrome picture on a colour monitor, or a colour picture on a monochrome monitor.
1050 The variable NBPLAN is set to the number of bytes to move on after each bit plane. The values of NBPLAN are :-Mode 0 NBPLAN = 8 (2^3) Mode 1 NBPLAN = 4 (2^2) Mode 2 NBPLAN = 2 (2^1) The variable OFSET is set to the number of bytes per bit plane, these are :-
Mode 0 OFSET = 20 (2^0) * 20
Mode 1 OFSET = 40 (2^1) * 20
Mode 2 OFSET = 80 (2^2) * 20
1060 Sets the colour palette, offset by two bytes from STRT, the colours are stored in the same way as in the sprite editor. (16 words per colour, ie 1 word a colour).
1070 Sets the variable MEM to the start of the compressed picture data (STRT + 34), and sets the variable SCANLINE to 0 (the top screen line).
1080 Start a REPEAT ... UNTIL 'condition' loop.
1090 Set the variable BUFFER to STRT+32098, the unused memory after the compressed picture is used as a temporary buffer area to store 160 bytes of uncompressed data. The variable BUFFER points to this area. Set the variable BYTE to zero, this contains the position in the buffer to store the next uncompressed data.
1100 Start a REPEAT ... UNTIL condition loop.
1110 Set variable DAT to the value at MEM in memory, and increment MEM by one (move to the text byte in memory).
1120 If the variable DAT is less than 128 then 'use the next DAT+1 bytes without any repetition'. Ie increment DAT by one, and set the variable FLAG to false.
1130 If the variable DAT is greater than 128 then 'use the next byte 257-DAT times'. Set the variable DAT to 257-DAT, and set the variable FLAG to true.
1140 Start a FOR ... NEXT loop, in this case from 1 to DAT.
1150 Store the value in memory at MEM, at BYTE in the buffer.
1160 Increment BYTE by one. (Move to the next position in the buffer).
1170 If the variable FLAG = false then increment MEM by one, move onto the next piece of compressed data. (This is when there is no repetition, tested for in line 1120).
1180 End the FOR ... NEXT loop.
1190 If the variable FLAG = true then increment MEM by one. (Move onto the next byte in memory if there is repetition, tested for in line 1130)
1200 End the REPEAT ... UNTIL loop if the variable BYTE = 160, ie 160 bytes per scanline irrespective of the current MODE.
1210 Start FOR ... NEXT loop, from 1 to number or bytes per bit plane.
1220 Start FOR ... NEXT loop, for each bit plane, the number of bit planes per mode are
Mode 0 No. bit planes = 3
Mode 1 No. bit planes = 2
Mode 2 No. bit planes = 1
1230 Poke the uncompressed data stored in memory as BUFFER into the destination screen memory. The destination is SCR+PLANE*2, PLANE*2 is used so that the destination is always on a word boundary.
(Continued Page 13)

Extending STOS

- (#2 of a few)

Last Issue, we looked at the basics of writing a STOS Interpreter extension. However, passing parameters to the extension commands and receiving them back from the functions was not covered. So you could only do commands like HELLO which don't exactly do much to show how powerfull the STOS extension system is!

Parameters are passed to the extension in quite a simple manner. All of them are passed in the stack (pointed to by A7 as usual) as longwords. Each parameter consists of 3 Longwords, which vary in their usage depending on whether the parameter is an Integer, real or string parameter. It is entirely up to your extension program to check for its syntax. When your extension is entered, the register D0 will contain the number of parameters on the stack (and so can be used to check that the number of parameters is correct). After you have stored the return address in **RETURN** (vitally important this), you can procede to pull the parameter values off the stack in **REVERSE** order. Stephen Hill recomends using the command **MOVEM.L (A7)+,D2-D4** to get each parameter off the stack, and I think its about the best way to do it, so I'll list the parameter types as they would be when pulled using this method: D2 is always a Byte value, and D3,D4 are always Longwords

D2 = **0 Integer**
D3 = Number
D4 = 0 (Not used, but still stored)
D2 = **$40 Real**
D3 = Top half of number
D4 = Bottom half of number
D2 = **$80 String**
D3 = Address of String
D4 = 0 (Again, not used but still stored)

The only thing regular machine coders will have to watch out for, is that the STOS strings are stored as a word for the string length and then the string itself, rather than the TOS format of a string followed by a zero byte to denote the end. Thus, when using the TOS traps with STOS strings, you will have to convert them. A usefull thing might be to create 3 subroutines to get the parameters for our extension. As these would be called with a **JSR** instruction (or a **BSR**), the stack would have the return address on the top whenever these routines are called. So the routine's first priority would be to pull this Longword off the stack and store it somewhere safe (say A0) with the command **MOVE.L (A7)+,A0** Then you would use the standard **MOVEM.L (A7)+,D2-D4** and you have your three values for the parameter.

You will probably want to check that you have the correct parameter, and thanks to the way Francois assigned the values 0,$40 and $80 to the 3 types, you can simply test them with the **TST.B D2** command, with Integers being checked for as zero (**BNE**), Reals being checked for as positive (**BLE**) and strings being checked for as positive (**BGE**). If your routine detects an error here, it should jump to a routine to display a TYPE MISMATCH error (See Error Messages later). After checking D2, you can return to the main program by JuMPing to the return address stored in A0 by **JMP (A0)**

An example routine to get an integer value (by far the most common and most simple):

```
GETINT: move.l (A7)+,A0 ; save the return address for later
        movem.l (A7)+,D2-D4 ; get the parameter data
        tst.b D2 ; check D2 for parameter type
        bne MISMATCH ; If not 0 then do a type mismatch error
```

Jmp (A0) ; return to the main program

Notice that you must do your own error checking. Upon finding an error, you can either use one of the standard STOS errors, or display your own error messages. The STOS error handler routine is part of the system routines available for use by your program. The vectors to these routines are stored in a table pointed to by the value in A0 when your extension is run. As you stored this value in **SYSTEM** at the start of the program, you can get it back at any point so that you can use it. The way to call a system routine is:

```
Move.l SYSTEM,A0
Move.l OFFSET(A0),A0
Jsr (A0)
```

the value of **OFFSET** depends upon which system call you want to use. The error display routine (Which stops the program, displays the error and stores its value in ERRN) is at **OFFSET=$14**, so to use this routine you would use

Move.l $14(A0),A0

This routine is called with the STOS error number in D0, so for a type mismatch error for the above input routine, D0 would have to equal 19. To display your own error message, you would use **OFFSET=$18**, where A2 contains the memory address of your error message in both English and French, and D0 contains the error number (To be put in ERRN). The only problem with using this routine is that it is unreliable, and tends to return the wrong error messages after you press UNDO twice to reset the screen display. So try to stick to using the standard STOS error messages if possible.

I'm going to try to get hold of a full list of the available system routines for STOS extensions (If you're reading this Francois, they would come in usefull !), but for now, here are two handy routines that check to see if a memory bank exists, and to convert a bank number to a screen address.

ADORBANK (Get a bank address) OFFSET=$88
Input: D3.L=Value.
Output:D3.L returns address of bank. If Bank number D3 doesn't exist, or D3 doesn't point to the start of a bank, an error occurs (automatically handled by STOS)
ADORSCRN (Get a screen address) OFFSET=$80
Input: D3.L=Value
Output:D3.L=Screen address : Exactly the same as ADORBANK, but it also checks to make sure that the bank is also reserved as a screen.

Now for the crunch test, another extension listing. This time we will add in a new command and a new function (both with parameters). The command will be called **PRNTER** (not **PRINTER** as this contains the reserved word **PRINT**) and requires one parameter which contains a bitmap for the printer settings. The function will be called **RANGE(A,B,C)** and will return the equivalent of **max(B,min(A,C))**. So if you wanted to make sure that A falls between 0 and 319 (usefull to keep things on the screen), you could use **A=RANGE(A,0,319)**.

N.B the data returned by a function is in the same format as the data entered to an extension except that you just leave the data in D2-D4 rather than pushing it back onto the stack.

```
* Not quite so useless extension
* Aaron Fothergill. STOS Club 1990
* adds the command PRNTER and the function RANGE(A,B,C) to STOS
* Initialise as normal
  bra INIT
* Tokens
  dc.b 128
TOKENS:
  dc.b "prnter",128
  dc.b "range",129
```

MACHINE CODE

```
    dc.b 0
    even
JUMP: dc.w 2 ; see Issue 8 for
details of how
    dc.l    PRNTER ; the tables
work.
  dc.l RANGE
MESSAGE: dc.b " 'Z' extension",0
  dc.b "extension 'Z'",0
  dc.b 0
  even
SYSTEM dc.l 0
RETURN dc.l 0
INIT: lea EXIT,a0
  lea COLDST,a1
  rts
COLDST: move.l a0,SYSTEM
  lea MESSAGE,a0
  lea WARM,a1
   lea TOKENS,a2
  lea JUMP,a3
WARM rts
*
* Now for the PRNTER command
PRNTER: move.l (a7)+,RETURN
  cmpi.w #1,d0
  bne SYNTAX
  jsr GETINT
  movem.l a0-a6,-(a7)
* routine uses Trap 14,33 to set
*printer configuration byte
  move.w d3,-(a7)
  move.w #33,-(a7)
  trap #14
  addq.l #4,a7
  movem.l (a7)+,a0-a6
  move.l RETURN,a0
  jmp (a0)
RANGE: move.l (a7)+,RETURN
  cmpi.w #3,d0
  bne SYNTAX
  jsr GETINT
  move.l d3,d0
  jsr GETINT
  move.l d3,d1
  jsr GETINT
  move.l d3,d2 ; now
* D2=A,D1=B,D0=C B<=A<=C
  cmp.l d1,d2
    blt TOOLOW ; A<B so make
A=B
TLBK cmp.l d2,d0
    blt TOOHIGH ; C<A so make
A=C
THBK move.l d2,d3
  move.l #0,d2
  move.l d2,d4
  move.l RETURN,a0
  jmp (a0)
TOOLOW move.w d1,d2
  bra TLBK
TOOHIGH move.w d0,d2
  bra THBK
*
* Support Subrotines
*
GETINT: move.l (a7)+,a0
  movem.l (a7)+,d2-d4
  tst.b d2
  bne MISMATCH
  jmp (a0)
SYNTAX: move.w #12,d0
  bra ERROR
MISMATCH: move.w #19,d0
ERROR: move.l SYSTEM,a0
move.l $14(a0),a0
  jmp (a0)
  dc.l 0
EXIT
```

(Continued on Page 21)

Absolute Beginners

#3 in a series of 1,000,000

I have been getting some feedback at last on this series, so I can now start tailoring it to help the right people. Learner programmers out there seem to be split into two groups:

Those who have programmed in Basic before but not STOS.

Those who have never programmed before.

So I am going to split the series into 2 seperate articles. The Absolute Beginners series will help those who really have never programmed before, and the Learners series will cover the aspects of programming a level up, for those of you just converting to STOS, or that have never attempted a major program before.

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

□ What am I doing ?

This should always be the first question you ask yourself before writing a program! A program is a sequence of commands that the computer will follow exactly, so you must make sure you get them exactly right! By deciding on what you are going to do in the program beforehand, and working out how exactly you are going to do it, you make your job a lot easier. For our dice game, the rules are:

The computer rolls a dice (a number between 1 and 6)
The player rolls a dice (another 1-6 number)
The highest roll wins

We also want to add a few embellishments to the game, such as being able to enter the player's name, and being able to play again.

Before writing the program, you might want to work it out in pseudo-code. This is where you write down what the program will have to do, without having to remember all the commands.

For our program we will have to:

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

As you can see, pseudo-code makes the program look a lot simpler. Believe it or not, the program really is that simple!

O.k, the first problem on our list is entering the player's name. We will have to store it somewhere, as we will need it later (to display it with his dice roll), so a **VARIABLE** must be used (see issue 7 for an explanation of what a **VARIABLE** is).

As the name is made up of a string of letters, a **STRING** variable has to be used (with a **$** after the variable name). We shall call the variable **NME$** .

The command required to get input from the user's keyboard and put it into a variable is called **INPUT**, and can be used to get one or more values from the user. The type of value depends upon the type of variables being input to (you can't input strings into a numeric variable for instance). You can also add a prompt to an **INPUT** command so that you can tell the user what to input. For our program the line will be:

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

This line will display the message:

Enter your name and press RETURN

with a flashing cursor. The user can then enter his/her name and upon pressing the

ABSOLUTE BEGINNERS

RETURN key, whatever they typed will be stored in the variable **NME$**.
Next we need to generate some random numbers for the dice throws. As we are only dealing with the numbers 1-6, we can use normal **INTEGER** variables to store the numbers in. The **FUNCTION =RND(n)** returns a value from 0-n, so to get a number from 1-6, we simply do **variable=RND(5)** and add 1 to it.
We will call the variables for the computer's throw **D1** and the player's throw **D2**, so the next 2 lines will be

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

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

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

Then we need to output the player's score:

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

Notice that on the player's display, we used the variable **NME$** as the first part of the print. This means that the user's name (if he typed it correctly) will be output as part of the message.
The only tricky part of the program is working out who wins. To do this, we need to use the **If...Then** statement. This works out the argument given to it, if it is true, then it will execute the series of commands after the **THEN** on the same line, otherwise it will go on to the next line (or whatever is after the optional **ELSE** statement)
First we will check to see if the computer has a higher score, this is true when **D1** is greater than **D2**. This can be checked by

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

Then we check to see if the player wins.

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

Finally we check for a draw..

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

Notice that > is greater than, < is less than, = is equal to (these can be mixed).
All we need to do now is see if the player wants another go, we can use another **INPUT** statement here and then check to see if the variable used contains "YES".

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

Notice on line 100 that you can use the **IF...Then** statement with string variables, and also the **GOTO** command, which makes the program **GOTO** the line specified (in this case 20).
If you've been typing in the program as you were reading this, just type **RUN** press **RETURN** and play the game. You've written your first 10-liner !

Richard Gales -PC? File decompressor

(Continued from page 8)

1240 End FOR ... NEXT loop
1250 Move destination to the next area of screen, and move onto the next uncompressed data in the buffer.
1260 End FOR ... NEXT loop
1270 Increment SCANLINE, move onto the next scanline.
1280 Repeat the above code until the final scanline is uncompressed. The final scanline is always 200 in every screen mode.
1290 End sub-routine.

Graphics Included !

Due to the STOS newsletter being entirely published on the ST now (it used to be Macintosh based), we can now include graphics and diagrams with your articles. They must be on disk in a picture format (Degas .PI3 preferably) and remember they will be printed in black and white !

STOS Paint Master

The most powerful art package ever written for STOS basic

Features include

a super low neochrome style display and easy to use icons.
Fast horizontal & vertical mirror,
zoom or reduce the screen to any size,
Pixel cut and paste, text, boxes, round-boxes, circles, ellipses, brushes, Lines, Colour Ramp, Load NEO, PI?, PC? and PAC files, magnify x2, x4 or x8.
STOS Paint Master also allows you to work in low or medium resolutions.
Includes a 512 colour mode, so you can display pictures in your own games.
Comes complete with free printer config & extension switcher accessories.
Send cheque/postal order for £14.95 (Overseas add £3.00) to

Stallion Software,

Devon

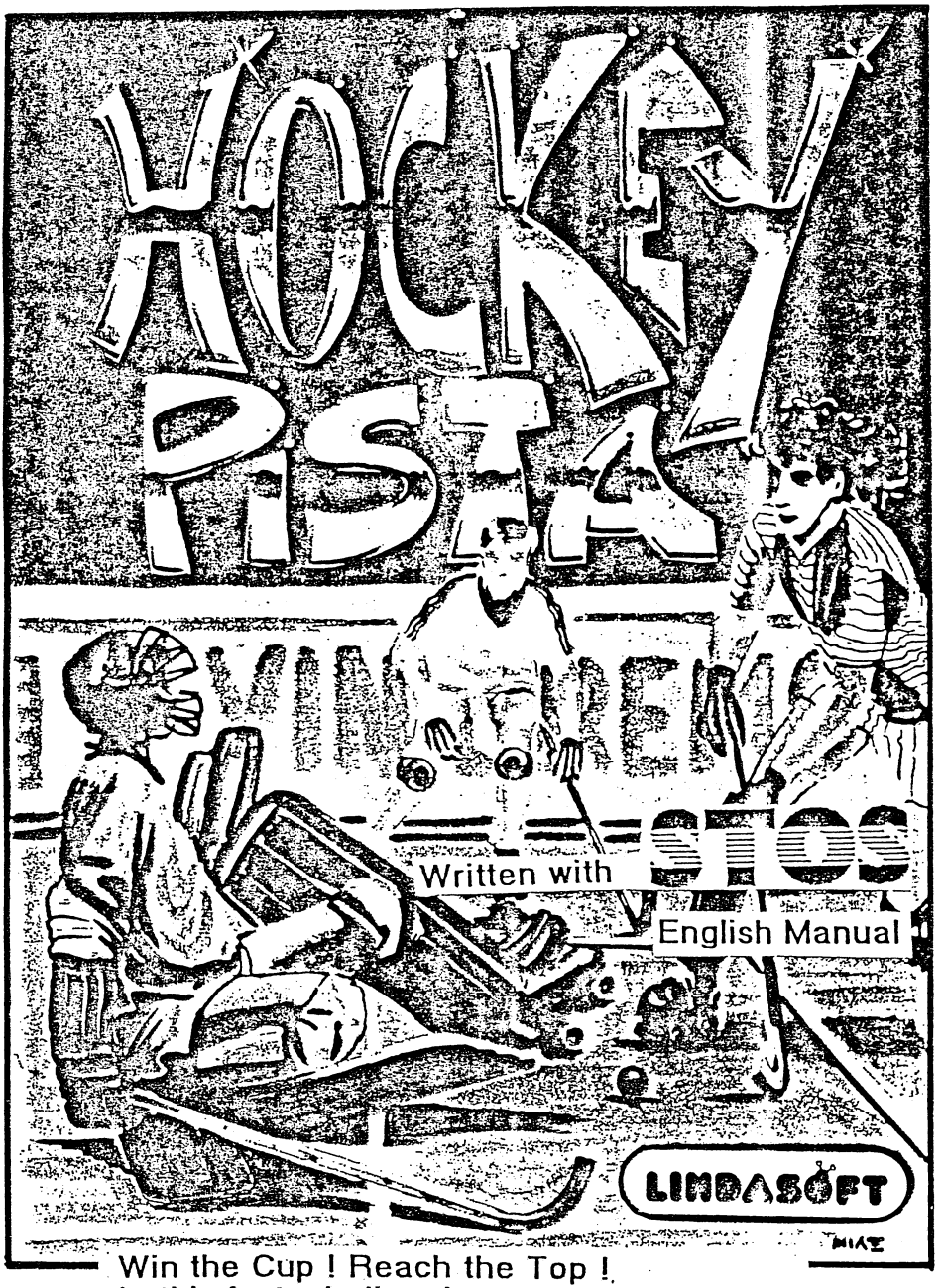
Making Cheques payable to Stallion Software.

Advertising in the Newsletter

How would you like to advertise your products to over 500 readers ? We are now running advertising in the STOS Newsletter, the cost of the advert pays for it's space, extra space for articles and a small sum towards paying for contributions to the magazine. The adverts get results, around 20% of readers buy a product advertised in the newsletter if it is good.
Here are the costs for full page and half page adverts.

Full Page	£50 Per Issue.
Double Page	£90 Per Issue.
Half Page	£30 Per Issue.

Adverts must be supplied either as an ASCII file on a disk, or as an A4 or A5 page suitable for black and white photocopying. Advertisements will only be accepted for genuine STOS related products, and anyone trying to advertise pirated software will be reported to FAST.
If you would like to advertise in the next issue of the newsletter, and would like more details, please contact me on the helpline. Also, if you are releasing a STOS-related product and would like it "plugged" and possibly reviewed in the Newsletter, send us a press release and review copy !



Win the Cup ! Reach the Top !
In this fast, challenging Sport Simulation.

Order directly to LINDASOFT - ITALY

(Refer to Mr. Doriano Benaglia)

Send cheque £.10 + £.4 (Shipping)

Scrolling Text

Why anyone should want to create a program that just scrolls text across the screen while playing a stolen soundtrack is beyond me ! But a lot of you out there are obsessed by writing demos. (about 1/2 of the calls on the helpline are from demo writers). Seing as the programming involved in writing a text scroller is about the right level, we are going to use it as the basis for this article for you "L" plated programmers.

Scrolling

The basic idea of scrolling is to take an area of screen, move it sideways, up or down, and then to place something new in the area it has been moved from (see diag.1). In STOS there are 2 ways of doing this:

1) Using **Def Scroll** to set up the scrolling area and **Screen Copy** or **Screen$** to copy over the blank

2) Using **Screen Copy** to do everything.

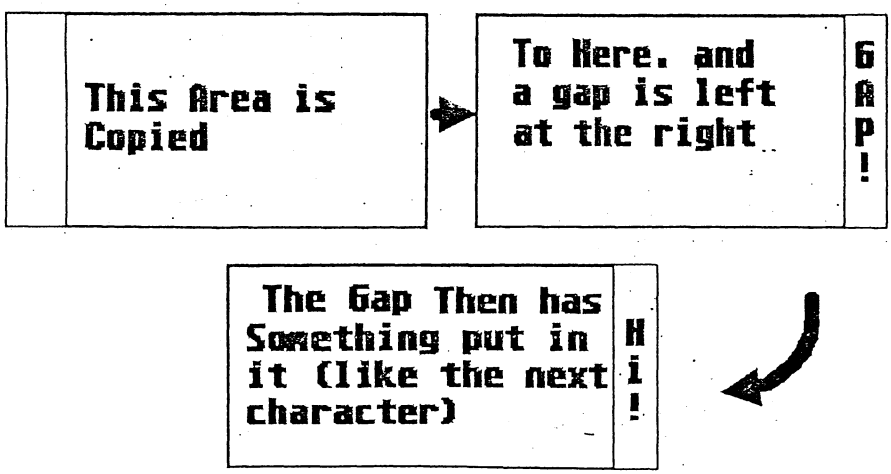
The later is both easier to understand and faster, plus it can be modified within the program by using variables for the parameters (if you wanted the scroll to change directions).

Diagram 1

We're going to try a very simple scroll, by scrolling a band on the screen and adding text to the right of the band, thus creating a scrolling message.

```
10 mode 0 : key off : curs off : hide on
20 reserve as screen 5 : Autoback off : Screen copy back to 5
30 rem the following line scrolls a band the width of the screen and 8 pixels high at 96 pixels down. Notice that it is done out of the way on screen 5 then copied to the physical screen.
40 screen copy 5,1,96,320,104 to 5,0,96
50 screen copy 5,0,96,312,104 to physic,0,96
60 rem add the text to the screen here
100 goto 30
```

O.k enter the program, but don't run it yet (it won't do anything anyway !). The next problem is how to add the text to the right of the screen, 1 pixel at a time. What we need to do is place 1 character at a time to the right of the scrolling zone (at pixel



co-ords 312,96) on screen 5. So we must set **Logic to 5** (so that all text and basic graphics functions work there). For the moment we will experiment with putting an "A" on the scroll line and repeating it. Note that with characters 8 pixels wide, scrolling at 1 pixel at a time, we must only place 1 character every 8 frames, this is where one of my favourite tricks comes in, the **MOD** function!

As explained in previous articles, if you have a variable and keep on adding 1 to it, then doing **variable=variable MOD N**, the variable will continuously cycle from 0 to N-1. This can be used to an advantage in our program, we simply count from 0-7, and whenever the counter goes back to zero (once every eight frames), we place the next character on the scroll line! Add these lines:

```
25 T=0
60 Inc T : T=T mod 8 : rem add 1 to T and make it cycle from 0-7
70 If T=0 then gosub 200 : rem once every eight frames place a character
200 logic=5 : rem point us at screen 5
210 locate xtext(312),ytext(96) : Print "A"; : rem place the character
220 return : rem go back to main loop
```

You can run the program now. You should end up with a series of smooth scrolling A's across the screen coming in from the right (Although they are slightly flickery, we can fix that later).

The next problem is how to display a whole message in the correct sequence of letters. Say for instance our message was stored in the string variable **MES$**, we would want to count along the characters and display them one at a time in our display routines. We could, for instance, use an integer variable as a pointer (using **MOD** again to make sure it never goes past the length of the text) to tell us which character to display, and then add one to it whenever a character is displayed. We could then use the **MID$()** function to pick out one character and display it. So:

```
26 P=0 : rem P is the pointer telling us which character to print next
27 MES$="Hello this is a scrolling message....." : Rem change this string to whatever you want in your message
210 A$=mid$(MES$,P+1,1) : rem get the next character to print
220 locate xtext(312),ytext(96) : print A$;
230 Inc P : P=P Mod len(MES$) : rem add 1 to P and loop it back to zero if it goes past the length of the message
240 return : rem go back to the main loop
```

Ta-da! We now have a scrolling message! Now to get rid of that annoying flicker. We shall use SCREEN SWAPPING (oh no not screen swapping!). Dead simple, we just need to change a couple of lines.

```
30 Logic=Back : rem this is needed before the screen swap
50 screen copy 5,0,96,312,104 to back,0,96
55 Screen swap : wait vbl
```

and voila, we have a totally smooth text scroller!

Just for kicks, do the following lines...

```
50 screen copy 5,0,76,312,124 to back,0,76+Y
51 Inc Q : Q=Q mod 360 : Y=20*sin(Q*pi/180.0)
```

next issue, we will make the text larger and display it over a scrolling background screen!

PROF SPECK O.D.D

The Art of Spagetti

By Prof A.Speck. O.D.D

If you have been given any formal programing lessons, or have learned your programing from a book, you will no doubt have been told that your programs should be structured, neatly arranged and covered in rem statements. If you have ever tried to type in one of these programs from a magazine or a book, you will no doubt have noticed the length of the program and become fed up of typing in all those rems !

Spagetti Coding is a popular artform amongst computer programmers, especially those using dialects of Basic which use line numbers. There are two forms of Spagetti Code, Practical and Freestyle. Practical Spagetti is a highly usefull form of programing, as it results in compact efficient code, wheras Freestyle Spagetti is a more artistic form, resulting in programs that are twice as long as they should be, but a lot more fun to look at. Look at these examples

☞Practical Spagetti

```
10 mode 0:key off:curs off:click off:dim a(10),b(10)
11fora=0to10:a(a)=rnd(100):b(a)=rnd(20):next a
```

☞Freestyle Spagetti

```
10 goto 1000

11 dim a(10),b(10):for a=0 to 10 12 goto 800
17 next a:end
500 curs off:click off:goto 11
800 a(a)=rnd(100):b(a)=rnd(20):goto 17
1000 mode 0:key off:goto 500
```

Both these programs do exactly the same thing, however the Practical Spagetti program is obviously more usefull, and the Freestyle Spagetti program is much more artistic. As Freestyle Spagetti is more of an artform than a usefull style of programing, we shall leave it in favour of PracticalSpagetti, which you will find usefull in most programs, especially where the program's length is restricted (such as in a 10-liner competition).

Where do the Rems go ?

The only problem some programers have with spagetti is remembering where things are in a program, and what they do. This usually results in the unethical use of rem statements in the program, which waste memory and make the listing more incomprehensible. However, advanced spagetti coders have developed an amazing piece of new technology, called the **High Resolution Double Density Fibre Based Fast Recall Memory System**, HRDDFBFRMS otherwise known as a pen and a piece of paper. By the wonderfull system of writing down notes while programing, you get to remember all those usefull details about program loops and variable usage, without cluttering the program with very unartistic rem statements. It is suggested that you experiment to find the best type and colour of paper and pen for use in your programing, as writing notes on a piece of green blotting paper with a magic marker might seem alright in theory, but in practice it is very impractical, especially if the magic marker is out of ink. Small ring bound notepads, and ballpoint type pens are recomended, although Freestyle Spagetti programers

have been known to use canvas and oil paint (Amiga spagetti coders use very large pieces of paper and blunt purple wax crayons).

☐ What can I Scrunch ?

There are practical limits to what can be compacted within a program. However for starters use the SEARCH command in STOS to find and remove all rem statements. Then you can procede to add lines together until they reach a suitable length. Usually a line length of 3 or 4 lines is suitable in medium resolution, although Freestylers might want to use low resolution to make the lines look longer (and they can then be done with more colours!).

Some statements however effectively end a programing line, thus limiting you in how you can compact the program. These are:

☞Rem : nasty, you don't want any of these in your programs

☞Goto : very usefull for Spagettifying a program

☞If .. Then : Only artistic if used with at least five And functions or 3 other embeded if's e.g

```
53 If (A=4 and b=4) or (((((c=8 or d=7) and q=2)))) then If z=3 goto 12
```

notice the artistic use of extra brackets in the above equation, a bit of Freestyle creeping in there !

☞Return : Try and get each of your subroutines onto 1 line, ending with a return statement.

☞End & Stop : Very terminal !

☞Data : try and avoid, as you can make the line much longer by directly loading the variables. E.g Instead of:

```
10 for a=0 to 20:read a(a):next a
20 data 1,10,53,64,97,.........,79
```

use

```
10 A(0)=1:a(1)=10:a(2)=53:a(3)=64:a(4)=97.....a(20)=79
```

In the best Spagetti programs, every line will end with one of the above statements.

As you can see there are various advantages of spagetti code over textbook style programing, and just because it looks nice, I have listed the various atributes of textbook,Practical Spagetti and Freestyle Spagetti in Table 1.

Feel free to try both styles of Spagetti coding. You might want to try and make one of your programs more effecient with Practical Spagetti, or you might want to hold a Freestyle Spagetti contest with your friends (or you might need new wallpaper). Either way, I'm sure you'll agree that Spagetti coding is much more usefull and a lot more fun. Remember

***** A good rem is in someone else's program ! *****

Professor Speck is a lecturer in applied Pseudophysics at the University of Advanced Gameswriting in Barnstaple. He is one of the leading exponents of experimental Freestyle Spagetti coding, and has won several awards and an Arts Council grant for his piece entitled "23567 ifs and several gotos". He is currently trying to break the his own world line length record of 15,032 statements on one line, using a Cray Mk2 Supercomputer and a specially modified version of STOS.

☞Technical Note:

You will no doubt have noticed that Spagetti is spelled differently from normal, this is because spagetti coders pride themselves on not having dictionaries ! In fact, most spagetti coders can't even spell dictionery !

HELP !

If you are stuck with your programming, and you need help, just phone the STOS Helpline on It is best if you phone between 1pm and 7pm as I'm most likely to be in then. Overseas members please note ! Check what time it is in the U.K before you phone, it isn't funny being woken up by the phone at 4a.m !

☞Table 1. Attributes of Programming Styles

❑ Textbook Programs

Very neat looking Impresses GCSE examiners

Incredibly long so they waste a lot of listing paper

Very memory wastefull

Not at all artistic They don't impress more junior programers (because it looks just like one of their GCSE programs)

❑ Practical Spagetti

Not exactly neat looking Makes GCSE examiners cringe (possibly an advantage!)

Much shorter but wider, so they use listing paper more economically

Very memory economical

Moderately artistic Impresses junior programers quite well

❑ Freestyle Spagetti

Looks like an exploded word-processor. Instantly fatal to GCSE examiners

Both longer and wider than both other styles of program. Makes great wallpaper!

Very wastefull of memory, just the excuse you needed to upgrade to 4 megs!

Incredibly artistic (especially if used as wallpaper) Impresses the hell out of junior programers, who will then spend years trying to trace all the loops and jumps in your program, before finally giving up and turning to flower arranging.

LETTERS

I have recently purchased STOS and I am creating a game for it. I have done my documentation and started to program the game, but I have got a few problems which I would like you to answer:

1) I have tried to do a pull down menu with the window in the middle of the screen. When I put in **menu on: border ,2** it disappears (Name of program & Window) and you have got a line at the top with the menu.

2) How do I add a save game option to my game ?

3) My next question is about the clock. I have started a clock going for my game and for some reason it carries on down a line, and when I put a **CLS** in it kept flickering and the clock goes in 2 second jumps. How can I smooth it out, and get it to go in seconds?

4) Finally, could you tell me how to go about asking a software company to publish my game ?

P.J Turner, Devon.

For a 2 option menu in the middle of the screen, you need to use an **"Alert"** box. Basically, you would have to save the screen area where you are going to draw the alert box (by **SCREEN COPY**ing it to another screen or saving it with **SCREEN$**), draw up the alert box and then use the **ZONE** command to set up zones for each button. Then you wait until the left mouse button is pressed, the line

: WHILE MOUSE KEY<>1 : WEND

should accomplish this. Once the mouse key has been pressed, return the zone that the mouse pointer is over with **Z=Zone(0)** If Z is less than 1 or Z is greater than the number of buttons, then none of the selections has been made, so you should go back to checking for the mouse pointer. Once a selection has been made, re-copy the saved section of the screen back to its orgional location.

To Save game positions, you will need to save **ALL** the variables relevant to the game. So if your game used the variables X & Y to store the players location, an array **E$()** to store equipment and so on, all these variables would have to be saved. The simplest way of saving them is to open a serial access file and output them to the disk that way. E.G

```
Open out #1,"SAVEGAME.DAT"
Print #1,X
Print #1,Y
For A=1 to 20
Print E$(A)
Next A
Close #1
To re-load the data, re-open the file and read it all in again:
Open in #1,"SAVEGAME.DAT"
Input #1,X
Input #1,Y
For A=1 to 20
Input #1,E$(A)
Next A
Close #1
```

The **TIME$** function only updates every 2 seconds. If you want a faster update, you can keep checking the **TIMER** variable, as this updates every 50th of a second. To avoid flicker, print or draw your clock to a seperate screen and then screen copy it to **BACK** and **PHYSIC**, or use screen swapping as described in issues 5-8 of the STOS Club Newsletter.

To persuade a company to sell your product:

1) Make it absolutely brilliant.

2) Find out who does their product evaluation

3) Send the product to them with all the documentation that they will need

4) If they suggest modifications, do them and keep on sending new versions until:

A) They give you lots of money ! or

B) They tell you in no uncertain terms that your game has about as much chance of being sold as an ice-lolly in the arctic ! Before sending any software off, make sure that it is completely bug free. Get as many