STOS BASIC ROUTINES

A collection of STOS Basic routines I programmed. Taken from Stosser, Power and ST+ diskzines

HIGH SCORE TABLE:

```
10 key off : cls

20 dim NAME$(10) : dim SC(10)

30 for X=1 to 10 : read NAME$(X) : next X

40 for X=1 to 10 : read SC(X) : next X

50 for X=1 to 10

60 print NAME$(X);" ";SC(X)

70 next X

80 print : input "Name:";A$ : input "Score:";B

85 for X=1 to 10

90 if B>SC(X) then swap NAME$(X),A$ : swap SC(X),B

100 next X : cls : goto 50

150 data "DEANO","MICK","JEFF","MIKE","STEVE","TOM","PHIL","PETE","JACK","ANDY"

160 data 1000,900,800,700,600,500,400,300,200,100
```

POKING CHARACTERS INTO A BANK:

```
10 reserve as work 5,400 : S5=start(5)

20 input "STRING:";A$ : LE=len(A$)

30 for X=1 to LE : poke S5+X,asc(mid$(A$,X,1)) : next X

40 for X=1 to LE : print chr$(peek(S5+X)); : wait 5 : next X
```

RANDOM NUMBERS IN AN ARRAY WITH NONE REPEATING:

```
10 cls

20 dim NUMB(12)
```

```
30 for LOOP=1 to 12
40 NUMB(LOOP)=0
50 next LOOP
60 for LOOP=1 to 12
70 repeat
80 EL=rnd(11)+1
90 until NUMB(EL)=0
100 NUMB(EL)=LOOP
110 next LOOP
120 for X=1 to 12 : print NUMB(X); : next X
```

RANDOM WORDS IN AN ARRAY WITH NONE REPEATING:

```
1 rem WRD$() holds the words to be placed in a random order
2 rem RWRD$() hold the same words but in a random sequence
100 key off
110 dim WRD$(12),RWRD$(12) : for W=1 to 12 : read WRD$(W) : next W
120 rem clear array
130 for W=1 to 12 : RWRD$(W)="" : next W
140 for W=1 to 12
150 R=rnd(11)+1 : if RWRD$(R)<>"" then 150
160 RWRD$(R)=WRD$(W)
170 next W
180 for W=1 to 12
190 print using "##";W;"  ";RWRD$(W)
200 next W
210 rem The 12 words, put anything here!
```

```
220 data "One","Two","Three","Four","Five","Six","Seven","Eight","Nine","Ten","Eleven","Twelve"
```

SIN WAVE LINES:

```
10 curs off : hide on : key off : mode 0

20 palette $0,$11,$577,$377,$177,$375,$573,$771,$750,$730,$700,$702,$703,$705,$727,$747

30 def scroll 1,64,16 to 256,200,0,8

40 logic=back : A=0 : C=1

50 X1=160+sin(A/21.6)*95

60 X2=160+sin(A/16.6)*95

70 Y1=58+cos(A/22.6)*40

80 Y2=58+cos(A/12.6)*40

90 ink C : C=(C mod 14)+2

100 draw X1,Y1 to X2,Y2

110 scroll 1 : screen swap

120 wait vbl : inc A

130 goto 50
```

SIN WAVE BOXES:

```
10 curs off : hide on : key off : mode 0

20 palette $0,$11,$577,$377,$177,$375,$573,$771,$750,$730,$700,$702,$703,$705,$727,$747

30 def scroll 1,64,16 to 256,200,0,8

40 logic=back : A=0 : C=1 : A=musauto(start(5),1,7858)

50 X1=160+sin(A/21.6)*95

60 X2=160+sin(A/16.6)*95
```

```
70 Y1=58+cos(A/22.6)*40

80 Y2=58+cos(A/12.6)*40

90 ink C : C=(C mod 14)+2

100 bar X1,Y1 to X2,Y2

110 scroll 1 : screen swap

120 wait vbl : inc A

125 if inkey$=" " then A=musauto(0,0,0) : stop

130 goto 50
```

FILL UP THE SCREEN WITH TILES

```
10 key off : hide on : curs off : flash off : mode 0

20 rem RESERVE BANK AS SCREEN AND ASSIGN LOGIC TO IT

30 reserve as screen 7 : logic=7

40 rem PLACE SPRITE TILE ON SCREEN AND MAKE A COPY OF IT

50 sprite 1,0,0,1 : put sprite 1 : wait vbl : sprite off

60 rem COPY THE TILE BLOCK TO A variable

70 T$=screen$(logic,0,0 to 16,16)

80 rem START OF LOOP

90 for X=0 to 330 step 16 : for Y=0 to 199 step 16

100 rem PLACE COPYS OF TILE AT CO-ORDINATES X AND Y

110 screen$(logic,X,Y)=T$

120 rem END LOOP

130 next Y : next X

140 rem SET SCREEN BACK TO NORMAL AND SHOW THE TILE PICTURE

150 logic=physic : screen copy 7 to physic
```

This routine fills the screen with copies of a sprite that has been copied on screen so it can be grabbed into a variable with SCREEN$. The sprite has to be a size of 16×16 pixels so the loop can make copies of it fit evenly onto the screen. The size could be changed but it must be in sizes of 16 so you could copy a 32×32 sprite block if you wish. This makes sure the loop can fit into the screen perfectly. Note the step size is that of the sprite size. I've also got it to build it up in a memory bank so you don't see it build up on the screen. This is quite fast but you could change it to grab a tile block from a screen in memory instead of the copied sprite block.

SIMPLE TEXT SCROLLER

```
10 rem SCROLLING TEXT LINE

20 :

30 rem SET SCREEN

40 key off : mode 0 : curs off : hide on

50 :

60 rem DEFINE SCROLLING AREA

70 Y=ygraphic(10) : Y1=ygraphic(11)

80 def scroll 1,0,Y to 320,Y1,-4,0

90 :

100 rem DEFINE TEXT TO SCROLL

110 T$="This is a simple scrolling routine which scrolls text in a variable across the screen……………………………. "

120 :

130 rem SCROLL THE TEXT

140 for L=1 to len(T$)

150 locate 39,10

160 print mid$(T$,L,1)

170 wait vbl : scroll 1

180 wait vbl : scroll 1
```

```
190 next L

200 rem START AGAIN

210 goto 130
```

It’s more or less straight forward. The use of the ygraphic command converts the text coordinates to graphic coordinates so the def scroll command knows where to place the scrolling zone. The above example will place the text at coordinates at 39,10 so the variable Y sets the top part of the scrolling zone downwards while Y1 sets the bottom part.

Y——————————————————————————-

THE SCROLLING ZONE FOR THE TEXT

Y1——————————————————————————

So Y equals y co-ordinate 10 whilst y1 equals the next line down which is 11. So the thing to remember is Y equals the y coordinate where you wish to position the text and Y1 equals the next text line. Try changing the values to see how it works. The loop does the scrolling. It works by placing one character at a time at the edge of the screen then scrolling it along by eight pixels then printing the second character at the same coordinate and moving that along. It continues this until the contents of the full string (T$) has been printed and scrolled. It's not very easy to explain so the best way to learn is to play around with the routine.

GET COLOUR HEX VALUES FROM A PICTURE IN MEMORY BANK

```
10 key off : curs off : hide : flash off : mode 0

20 get palette (5)

30 for X=0 to 15 : print hex$(colour(X)) : next X
```

CHANGING PALETTE VALUES OF PICTURES DRAWN WITH STOS

```
10 key off : hide : flash off : curs off : mode 0

20 for X=0 to 15 step 10 : paper x : bar X,10 to X+10 : next X

30 rem Coloured boxes on-screen, now lets change colours

40 wait key
```

50 palette $0,$777,$756,$676….etc: Rem put 16 hex numbers on a line and

change one or two

60 wait key

70 colour 1,$0 : rem change colour one to black

This method won’t work with a picture saved from an art package. What I think you need to do is poke the new hex value into the screen bank. The first 32 bytes of the bank is the screen palette.

UNPACK AN MBK SCREEN

STOS has two commands for packing and unpacking pictures. These are PACK and UNPACK. The MBK file has been packed with the STOS compact accessory so you need to use the unpack command in your routine to unpack it like so.

10 key off : hide : curs off : flash off : mode 0

20 load”picture.mbk”,5

30 unpack 5,back

40 screen copy back to physic

So, in other words, line 20 loads the packed picture into bank five. For MBK files you don’t usually need to reserve a bank first. The unpack command then unpacks the picture to the background screen then the whole screen is copied to the main physic screen. Note you can also unpack a screen into a bank.

NICE FADE EFFECT

10 key off : hide on : flash off : curs off : mode 0

20 rem first set all colours to white

30 wait 10

40 fade 5,$777,$777,$777,$777,$777,$777,$777,$777,$777,$777,$777,$777,$777,

$777,$777,$777,$777

50 rem wait for fade to happen

60 wait 80

70 rem now fade colours back to the present palette

80 fade 5,$523,$777,$0,$123,$232,…etc

FIND WHICH KEYS ARE ASCII CODES AND WHICH ARE SCANCODES

10 key off : hide on: curs off

20 print”press a key”

30 c$=inkey

40 if c$=”” then goto 30

50 if not(scancode) then print”This is an ascii character” else the scancode

for this key is “;scancode

60 goto 20

PRINT CONTENTS OF A FOLDER

10 key off : hide on

20 dir$=”DATA”: rem name of the folder

30 print all files to the screen

40 P$=”.“: N$=dir first$(P$-1): if N$=”” then end

50 print N$

60 repeat

70 N$=dir next$

80 print N$

90 until N$=””

100 dir$=”A:”: rem close folder and default back to root directory

110 rem print contents of folder on printer

120 dir$=”DATA”

130 ldir

```
140 dir$=”A:”
```

GET RID OF THE MISSING LINK REGISTRATION REMINDER MESSAGE

In the full version of the missing link you get a third extension (EXS) which contains a command “mostly harmless” to gets rid of that message. Type out this routine.

```
10 A=mostly harmless (1,2,3,4,5)

20 put key “NEW”
```

Save this in the root directory (not in a folder) of your STOS disk under the name of ‘autoexec.bas’. Now when you load STOS this file will run, a message will appear saying ‘You got it’ and the NEW command pops up. Press return and the message has gone.

PASSWORD ROUTINE THAT STORES IN A VARIABLE

```
10 key off : hide on : curs off : mode 0

20 locate 0,10 : centre”Please enter the password.”

30 A$=input$(5) : rem password is five characters long

40 if A$=”POWER” or A$=”power” then locate 0,12:centre “Okay, loading.” else locate 0,12: centre”That is not the password, try again.” : goto 30
```

SCREEN UNPACKING TIP

Normally when you unpack a screen from a bank, the screen palette changes. The method to fix it is to use the missing link “floodpal” command like this:

```
10 key off : flash off : curs off: mode 0

20 reserve as screen 5 : unpack 4,5 : floodpal 0 : wait vbl

30 get palette (5) : screen copy 5 to back : screen copy back to logic
```

The wait vbl is important in this routine.

Thanks to Tony Greenwood for this tip.

HAVE SOMETHING HAPPEN AFTER TWO HOURS

```
10 time$=”10:00:00?
```

```
20 repeat

30 until time$=”12:00:00?

40 print “Two hours in real-time have passed.”
```

SIMPLE SPELL CHECKER

How this routine works are to hold a list of words in arrays and check each word of the document against the ones in the arrays. In the spell checking part of a WP, all words are stored in alphabetical order, so first we can store a list of words beginning with A in a data statement. We can then use the SORT command to put them in order then the MATCH command to check each word in the document against the words in the array. Assuming each word of the document is held in an array then this routine does the checking.

```
10 rem A$ array holds all words in doc, B$ array holds words to check

against

20 dim B$(3)

30 for X=1 to 10 : read B$(X) : next X

40 sort B$

50 rem Check all words in doc against words in B$

60 inc W : W$=A$(W) : POS=match(B$(0),W$)

70 if POS<0 then goto 90

80 if W<10 then goto 60

90 rem WORD NOT FOUND

100 for X=0 to 2 : print B$(X) : next X

110 data”love”,”lovely”,”lover”
```

What happens here, is that the SORT command puts the words you want to check against the doc in alphabetical order, then the W$ variable gets each word of the document and checks it against the sorted word array (B$). If POS is less than nought then the routine has found a word it doesn’t understand and will print out the list of words it does know. What you can do for speed is just list the words beginning with the first letter of the word that the routine couldn’t find. Like this.

```
120 rem A$ holds all words beginning with A

130 if left$(W$,1)=”a” then goto 140

140 for X=0 to 2 : print B$(X) : next X
```

RPG AND ADVENTURE GAME EXITS ROUTINES

The quickest way of reading things like exits and such in a maze of rooms is to store all the info in arrays, then check which room the player is in and set zones in that room.

```
10 key off : curs off : mode 0

20 dim MAP(5,4) : dim XZ1(5,4),YZ1(5,4),XZ2(5,4),YZ2(5,4) : ROOM=1

25 rem SET UP MAP EXIT VALUES

30 for X=1 to 5 : for Y=1 to 4

40 read MAP(X,Y)

50 next Y : next X

52 rem SET UP ZONE CO-ORDINATES FOR EACH ROOM

55 for X=1 to 5 : for Y=1 to 4

66 read A,B,C,D

70 XZ1(X,Y)=A : YZ1(X,Y)=B : XZ2(X,Y)=C : YZ2(X,Y)=D

75 next Y : next X

76 rem DRAW EXIT BOXES AND INFO OF WHERE EACH ONE GOES

80 home : print “ROOM”;ROOM : locate 0,6

90 for X=1 to 4

100 box XZ1(ROOM,X),YZ1(ROOM,X) to XZ2(ROOM,X),YZ2(ROOM,X)

110 set zone X,XZ1(ROOM,X), YZ1(ROOM,X) to XZ2(ROOM,X), YZ2(ROOM,X)

115 print “EXIT”;X;” GOES TO ROOM”;MAP(ROOM,X)

120 next X

125 rem WAIT FOR PLAYER TO CHOOSE A ZONE
```

```
130 repeat

140 EXIT=zone(0)

150 until EXIT<>0 and mouse key=1

155 rem SET ROOM VALUE TO NEW ROOM

160 ROOM=MAP(ROOM,EXIT) : goto 80

1000 rem EXIT VALUES FOR EACH OF THE FIVE ROOMS IN MAZE

1010 data 2,3,4,5

1020 data 1,2,3,4

1030 data 2,1,3,4

1040 data 3,2,1,5

1050 data 1,2,3,4

2000 rem ZONE CO-ORDINATES FOR ROOM ONE

2010 data 10,20,20,40

2020 data 30,20,40,40

2030 data 50,20,60,40

2040 data 70,20,80,40

2060 rem ZONE CO-ORDINATES FOR ROOM TWO

2070 data 10,20,20,40

2080 data 30,20,40,40

2090 data 50,20,60,40

2100 data 70,20,80,40

2120 rem ZONE CO-ORDINATES FOR ROOM THREE

2130 data 10,20,20,40

2140 data 30,20,40,40

2150 data 50,20,60,40

2160 data 70,20,80,40
```

```
2170 rem ZONE CO-ORDINATES FOR ROOM FOUR

2180 data 10,20,20,40

2190 data 30,20,40,40

2200 data 50,20,60,40

2210 data 70,20,80,40

2220 rem ZONE CO-ORDINATES FOR ROOM FIVE

2230 data 10,20,20,40

2240 data 30,20,40,40

2250 data 50,20,60,40

2260 data 70,20,80,40
```

In this routine, we have five rooms in the maze, each with four exits which all lead to other parts of the maze. The MAP array works like this: the variable ROOM holds the number of the player's position in the maze, in other words, the room number. The first set of data statements allow us to specify which exit leads to which room. For example….

```
10 read MAP(1,1),MAP(1,2),MAP(1,3),MAP(1,4)

20 data 2,3,4,5
```

This means that exit one leads to room two, exit two leads to room three and so on. When you run this routine you will see four boxes on the screen, representing four zones, or exits in this case. Clicking the left mouse key

in a zone will set the variable ROOM to the new destination room. With this method, you can easily tell where you are just by reading the ROOM variable. With this, you can call up the part of the maze you want.

```
100 screen$(logic,16,0)=R$(ROOM)
```

The arrays XZ1, YZ1, XZ2, and YZ2 hold the coordinates of the exit boxes on the screen. The format goes:

```
XZ1(ROOM,EXIT)
```

So, we can set up four zones in the present room using the SET ZONE command then check which zone the player chooses. The exit zones could be four arrows pointing in four directions. Using this method, you can check what’s on-screen in this method. For example, if the player clicked on a baddie to fight him, then you could say that the baddie is in room three and is in zone two, and check like this.

```
10 repeat

20 CH=zone(0) : wait vbl

30 until CH<>0 and mouse key=1

40 if ROOM=3 and CH=2 then gosub 1000
```

Where line 1000 onwards holds the routine for fighting the baddie.

BANK STACKING

Think of a large box, this allows us to put more than one item into it. It’s the same with banks, we can set the size of it then load the files into it one after another. This method with stack PAC pictures.

First make a note of the length of each PAC file then add them all together, then add about twenty bytes and make it an even number. You then choose a bank and reserve it to this size.

```
10 reserve as data 5,80000
```

Note reserving a work bank makes it a temporary bank while reserving a data bank means it can be saved along with your program.

Load the first file in like this.

```
20 load”pic1.pac”,start(5)
```

Now the next thing to do is load the next picture into the position where the last picture ends. So, get the length of the first file and make it into an even number. For example, if the file is 1787 bytes long then call it 1790 bytes long. We can now load the second file in, in front of the first file just like this.

```
30 load”pic2.pac”,start(5)+1790
```

After this, we just take the length of the pictures already loaded and load the next picture in. For example, let’s say our pictures are like this.

PIC1.PAC LENGTH 1787 ROUNDED TO EVEN 1790

PIC2.PAC LENGTH 2136 ROUNDED TO EVEN 2140

We take these two values and add them together, which makes 3930. If the value is an odd number then you must make it even. We can then load this file in.

```
30 load “pic3.pac”,start(5)+3930
```

Do the same with the other files already loaded, add them together and round them up to even numbers to find where to load the next picture in. To get at them, all you have to do is this.

```
40 unpack 5 : rem unpack the first picture

50 unpack start(5)+3930 : rem unpack picture three
```

There is an easier method if you have the missing link extension. This has two commands called BANK LOAD and BANK COPY. Use the MAKEBANK program on the source disk to load all files and save them as an FBANK with the extension BNK. You can then load and use them like this. PICNO is Picture Number.

```
10 mode 0 : key off : flash off : curs off

20 PICNO=1

30 reserve as work 5,80000

40 bload”pics.bnk”,5

50 reserve as work 6,9000

60 bank copy start(5),start(6),PICNO

70 unpack 6
```

This copies the specified picture from the stacked bank into bank six and unpacks it to the screen. You can use this method with the smaller files but it tends to corrupt some larger ones. The first method however should work with all files.

In case you were wondering, the BANK LOAD command allows you to load a file from a stacked bank on a disk.