10.7.1990

INAGE PROUDLY PRESENTS ANOTHER STUNNING RELEASE;

```
       AAAA         Minn      nnnn     00000000      SSSSSS
      AAAAAA        nnnnnn   nnnnnn   0000    0000    SSSS  SSSS
     AAAAAAAA       MliMIIMMMMMMMIIMMil  OOOO    OOOO  SSSS    SSS
    AAAA  AAAA      nnnn nnnnri nnnn  0000    0000   SSSSSSS
   AAAA      AAAA   nnnn      nnnn     00000000      SSSSSSSSS
```

T H E   G R E A T 0 R

ThanxtoMr.. Spaghe11o/Accessionforthetypin' work!

!(Usinq CyqnusEd Professional Release 11)

Without the great help of

Stranger/1maqe

you woudn't; have this Fi1e/Print. in your hands. He de1ivered me the original! manual, and promised to spread the Final Version of this file all around the world.

-....Try the NO-1 BBS in Finland, give a call to IMAGE HQ -...
— Pirates Cove at: +358-0-802 4389 (HST DS, 121MB, 24H) —

How was it all jdone? •
=====================

AMOS Basic was designed and progrqammed by François Lionet„ His clever ideas and inspirational work have produced what we feel to be by far the best high-level programming language available on the amiga to date.

AMOS was developed using the following programs;

DEVPAC II Assembler - HiSoft
Deluxe paint III - Electronic Arts
Pix Mate - Progressive Peripherals 8: Software
Cross-Dos - Consultron
Mini Office Professional Communications - Database Software

Mandarin Software would like to thank the following people for their kind help during the development of AMOSs

A3.listair Brirnble, Aaron and Adam Fothergill of Shadow Software,, Peter Hickman, Rico Holmes, Commodre UK for the international keyboard 1ayouts (an the Amiga)., Commodore France for the help with the A1000 problem,, 17-Bit Software for samples and demos, Martyn Brown for fonts and support., Virus Free? PD for Soundtracker, Simon Cook for his constructive comments and bugfinding, Lee, Alex, a11 other AMOS developers for their kind, help and all of you who have waited patienly for this software. We hope, like us, you feel it was well worth the wait.

This manual (the original ,, not this ASCII file) was written using WriteNow on the Apple Macintosh and paged up with Page Maker.

CONTENTS

1: INTRODUCTION

WELCOME to the exciting world of AMOS - The Creator! As you know, the Amiga is a truly amazing computer. For the first time, all that power is at your fingertips.

In Septemper 1988, Mandarin Software released STOS Basic for ST. This made history as the first programming language to reach number one in the ST Gallup games charts! Now STOS has been rewritten from the gound up to produce AMOS Basic for the Amiga. AMOS Basic includes a vast range of over 500 commands - many of which are staggeringly powerful. You can, for instance, bounce a screen,, or animate a sprite using just asing1eBasicinstruction.

AMOS is not just another version of Basic - it's a dedicated games creation system which comes with its own built-in Animation Language (AMAL). AMAL programs are executed 50 times a second using a powerful interrupt system. They can be used to generate anything from the attack waves in an arcade game, to a silky-smooth hardware scrolling efect. At the same time, your Basic program can be doing something completely different!

Whatever you knowledge of programming, AMOS has something to offer you. If you have newer written a game before, the prospect of creating your first game may be quite daunting. But do bear in mind that many of the all-time classics *ara-* uncomplicated programs with one or two original features -- just look at Tetris for example. The strength of your game will depend on the quality of your ideas, and not just your programming skill. With a little help from AMOS, you'll be able to produce professional-looking games with just a fraction of the normal effort. All you really need is imagination.

If you've written a game in AMOS basic,, don't keep it to yourself. Mandarin Software is very keen to publish any program written using AMOS. Don't worry if your programming is a little rough. If your ideas are good enough, you could have a real future as a professional games writer. So please send us your programs,. Mandarin would also be delighted to hear your comments or suggestions for the AMOS system,. Several features in AMOS were taken directly from the ideas which were sent to us from existing STOS users. Address your correspondence for the attention of Richard Manner, Development Manager, Mandarin Software, Adlington Park, Ad ling ton, Macclesfield SK10 4NP,

21! GETTING STARTED

AMOS Basic is a truly remarkable package, capable of creating games which were previously beyond your wildest dreams., All this powerful •features which make the Amiga so irresistible have been incorporated into this amazing system. With help of AMOS Basic you can develop programs which would tax the skills of even the most expert assembly language programmer.

You can for instance, effortlessly animate up to 56 hardware sprites simultaneously! This is a real achievement, especially when you consider that the Amiga's hardware only actually provides you with eight.

If you need even more action on the screen, you can use the Amiga's blitter chip as well. Blitter objects can be created in any graphics mode you like, including HAM) The only limit to the number of bobs on the screen is the amount of available memory.

Any combination of the Amiga's graphics modes can be displayed on the screen at once. Hardware scrolling isn't jsut possible., it's easy! There's a built-in SCREEN OFFSET command which allows you to perform the entire process directly.

In fact, the only hard part of AMOS Basic is knowing where to start! AMOS supports over 500 Basic commands, and if you're never used Basic before, you may feel a little overawed by the sheer scale of this system,. When you're in unfamiliar territory,, it's always useful to have a *GUIDE* (Thanks to me!, Mr.Spaghetto ;••••) to show you around and point out some of the notable landmarks,, That's the purpose of this chapter.

Backup AMOS Know* !

Before continuing however, it's vital that you back up the entire AMOS Basic package on fresh discs. This will safeguard your copy of AMOS against accidental mistakes. You'll now be able to play around with the system as much as you like, without, the risk of destroying something important.

If the worst comes to the worst,, we at Mandarin will be happy to replace your disc for a nominal handing charge. But you'll obviously be deprived of AMOS Basic while it's being re-duplicated,

The installation procedure varies depending on your precies set-up, but it can usually be accomplished in a matter of minutes,.

How to backup?

If you have got this Ascii file into your hands,, you propably also have some cool copy-prog,, for example? X-Copy,, I)--Copy or other...

Place the originals into a safe place and use the fresh copies now.

Loading AMOS Basic

As you might expect. AMOS Basic can be executed in A uariety of different ways. You can, for instance, load AMOS directly from the Workbench by selecting its icon with the left mouse button. Once you've entered AMOS in this way., you will be able to flick back and foth to the Workbench by pressing the Amiga and A keys from the keyboard.

In practive however, the Workbench consumes valuable memory which would' be better used to hold your Basic programs. So if you're a serious user, you'll propably prefer to boot up AMOS as part of your normal start-up sequence. This will allow you to achieve the maximum possible results from the AMOS system.

To load AMOS Basics

* Turn off your Amiga and wait for about ten sees.
* Place a backup of the AMOS program disc (disc 1) into DFOs
* Now switch on your Amiga. AMOS will load into memory automatically.,
% Hit a key to remove the information box and thus enter the AMOS system.

AMOS tutorial
=============

The first thing you'll see when you enter AMOS Basic is the editor window. This is extremely easy to use, and if you've a little previous experience with computers it should be self-explanatory. Feel free to experiment as much as you like. The AMOS editor is quite intelligent, and you *Are* unlikely to make any serious mistakes.

Now you've seen the editor window, It's time to explore some of the, features that make AMOS Basic: really stand out from the crowd.

Loadingaprogram

We'll start off by showing you how you can load one of the terrific games from the AMOS data disc. We'll take the Number Leap game as an examples

* Insert the AflOS_DATA disc into drive DFOs
* Hold down an Amiga key on the keyboard and press "L",, This will bring up a standard file selector on the screen.
* Click on the disc drice label DFO to inform AMOS that you have changed the disc
t At the centre of the file selector there will be a list of programs which can be loaded into AMOS Basic.
* To select the Number Leap program, just position the mouse pointer over the file:

  Mumber_j_eap, AMOS

  The file you. have chosen will be highlighted accordingly.

* Once you've chosen your file, you can load it by clicking twice on the left mouse button. Your game will now be entered from the AMOS DATA disc and you will be returned to the original editor-screen. The contents af this window wi 11 be updated to display your new program listing.

* You can run this program by selecting the RUN button from the main menu area (or hit Fl if you're feeling lazy).

The editor screen will now disappear completely and Number_Leap will be executed in front of your eyes. After you've played with this game to your satisfaction, you can exit to AMOS Basic by pressing the CTRL and C simultaneously.

CTRL+C provides an effective way of breaking into the vast majority of AMOS programs. It can be disabled from within your program using a BREAK OFF command for extra security. When the program has been broken into yu can flick straight back to the editor by pressing the Spacebar key from the keyboard.

Deleting a program

Now that we've finished with the Number Leap program, we can erase it from memory with the NEW command. You won't find this option on the main menu, as it's been placed in a separate SYSTEM menu. This can be brought into view by moving the mouse pointer over the menu window and holding down the right mouse button.

To delete a programs

* Ensure the mouse pointer is over is over the menu area.
* Hold the Right mouse button down to bring up the SYSTEM menu,.
* While the button is depressed, move the pointer over the NEW option and select it with the Left mouse key. Alternatively, you can execute this option directly from the keyboard by pressing Shift+F9.
* Type Y to confirm the operation or N to abort.
* If the current program hasn't been saved, you'll been asked whether to store it onto the disc. If you select the YES option, you'll be presented with an AMOS file selector,, Otherwise your program will be totally erased.

Direct mode

We'll now have a quick look at the direct mode. This forms the centre of the AMOS Basic package and allows you to experiment with your routines and immediately observe, the effects.

It's important to recognize that all the screens, sprites, and music: defined in your program are completely separate from the Editor window. So no matter what you do in direct mode, you'll be able to return to your listing with just a single keypress,,

* Enter direct mode by pressing ESCape,, The editor window will slide away and you'll be presented with the main program display.

Towards the bottom of this area will be a small screen which can be used to enter your direct mode commands. Try typing the following line, pressing Return to "execute".

```
        Print "Your name"
```

Insert your name between the quotes to print your names on the Amiga's screen. Now press the UP and DOWN arrows from the keyboard to move the window around the display area. As you can see, the Direct mode window is totally independent of the main program screen,,

Animation i

So much for the Direct (node. Let's experiment with some of the AMOS load

to load a set of sprite images into memory. Stay in direct mode and enter the indented lines in bold as you come to them,.

Listing the sprite files

We'll begin by listing all the available sprite files to the Amiga's screen.

* Ensure that the **AMOS** DATA disc is still in the DFO:
* Display the disc file directory with the lines

```
Dir "AMOS_DATAsSprites/"
```

This will display the sprite files we've supplied on the AMOS data disc. These files contain all the images which are used in the various example programs. You can create your own images using the Sprite definer accessory on the ANOS Program disc.

The sprite definer incorporates a host of powerful drawing features which make it extremely easy to generate professional-quality animation sequences in your games.

Loading a sprite file

We can now load these sprites using the LOAD command,, The sprites will load into a special memory bank so don't except to see any sprites to appear yet! Let's enter the sprites used by the Number Leap game with the fo11o wing co mma nds

```
Load "AHOS_.DATAsSprites/Frog_Sprites.abk"
```

If you make a mistake, hit Fl to get your previous line,, This line can be then edited using the normal cursor keys and may be re-executed by pressing Return.

Now let's also load up a music file using a similar load command:

```
Load "AMGS.JDATA_Sprites/Funkey.abk"
```

In order to check whether the sprites and music have been succesfully loaded into memory, we'll call up the LISTBAMK instruction like so:

```
Listbank
```

This prints a line like::

```
1 - Sprites      S;*0682B0  L:000040
3 - Music         Ss*043878  LsOOBlFE
```

Don't worry if the numbers do not correspond as they will change depending on the available memory. The number of sprites we've just loaded can be returned direc11y with the LENGTH function„

```
Print Length(l)
```
(results 64)

Setting the sprite colours

Each set of sprite images has its own set of colour values stored on the disc. Since these can be very different from your current screen colours, it's useful to be able? to GRAB the colours from the sprite bank and copy thein in to an existinq screen,, This can be accomp 1 ished with the GET SPRITE PALETTE command. Enter the lines

```
        Get Sprite Palette
```

Ail the colours in the main program screen will change immediately, but the direct mode window will be completely unaffected because it's been assigned its own separate list of colour values by the AMOS system.

Displaying a sprite

Sprites can be displayed anywhere on the screen using a simple AMOS Basicsp▸itecommand.Here'sanexamp1es

```
        Sprite 8,129,50,62
```

Animating a sprite

Let's animate this object using The "Ailos Animation Language". AMAL is a unique animation system which can be used to move or animate your objects at incredible speed.

Note that when you're entering the following example programs,, it's essential to type each line fcexactly* as its appreas in the listing, as otherwise you *nM\y* get an unexpected syntax error,,

```
        Sprite 8,129,150,62
        Amal 8,"Aniift'0,(62,5)(63,5)(64,5);" s Amal On
```

The program above animates a small duck on the screen„ Whilst it's being manupulated,, the sprite en be moved around using the SPRITE command. Examples

```
        Sprite 8,300,50,
```

Moving a sprite

Now for some movement!

```
        Sprite 8,129,150,62 : A* = "Anim 0,(62,5)(63,5)(64,5);"
        A*=A*+"Loop: Move 320,0,100; Move --320,0,, .1.00; Jump Loop"
        Amal 8,A* : Amal On
```

This programs animates the duck and moves it back and forth across the screen, using .just three1ines!

Although the instructions between the quotes may look like Basic, they're actually written in AMAL. All AMAL programs *Are* executed 50 times a second and they can be exploited to produce silky smooth animation effects independently of your Basic programs.

Just to prove how amazing AMAL really is,, hit ESC to jump back to the Basic editor. After a few moments;, return to direct mode. Your sprite will still be bouncing accross the screen as if nothing had happened!

Music maestro!

For a finale, let's play the music! Ensure you're still in direct mode, **turn** UD the uolumra *an* ynuf *man i.* +.*a* I" .arid stipt -lh«> [illegible]
MUSIC command like so:

```
        Music 1
```

By the way, you can stop the music: with the commands

```
        Music Off
```

The joyrney continues

Hopefully, you'll now have a pretty good idea of what AMOS Basic can achieve. But so fat we've only looked at a tiny fraction of AMOS Basic's power,, As you experiment with the AMOS package, you'll quickly discover a whole new world, full of exciting possibilities..

AMOS Basic can't, of course, transform yau into an expert games programmer overnight. Like any programming language, it does take a little time to familiarise yourself with the entire repertoire of commands. We'll therefore end.this section with a few guidelines to help you on your way.

Hints and tips

% The best way to learn about AMOS is to create small programs to animate sprites,, scroll screens or generate hi-score tables. Once you've created a little confidence,, you'll then be able to incorporate these routines into an actual game.

% Don't be overawed by the sheer size of the AMOS Basic language. In practive, you can achieve terrific effects with only a tiny fraction of the 500 or so commands available from AMOS, Start by mastering just a couple of instructions such as SPRITE and BOB,, and then work slowly through the various sections. As you progress, you'll gradually build-up a detailed knowledge of the AMOS system.

% Although we've attempted to make this package as easy to use as possible, a thorough groundging of the general principles of Basic programming is invaluable. If you're new to Basic, you may find it helpful to purchase an introductory text such as "Alcock's Illustrating Basic. (Cambridge University Press.)

* P1an your games car ef u11y on pape r„ 11" s amazing how man y prob1ems 10 can be completely avoided at the early design stages. Never attempt to tackle really large projects without prior preparation. It's the easiest way to get permanently lost.

* When you're writing a game, try to concentrate on the quality of the game play rather than the special effects. The graphics and music: can always be added later if the idea's are good enough.

3: THE EDITOR

The AMOS editor provides you with a massive range of editing facilities,, Wot only is it exceptionally powerful, but it's also delightfully easy to use. All commands can be executed either directly from the screen,, or via an impressive range of simple keyboard alternatives. It's so friendly in fact, that if you've a little experience with computers,, you'll propably be able to use it straight out of the box.

One of the most exciting features of this sytem, is that the listing is displayed completely separately from your main program screen. So you can instantly flick from your program display to the editor window using a single keypress (ESCape). ......

If you've plenty of memory,, it's also possible to load several programs in AMOS Basic at a time. Each program can be edited totally independently, and it's possible to efforlessly switch between the various programs in memory by pressing just two keys from the editor.

The first thing you see after AMOS has loaded into memory is a standard credit screen. Applause applause! Press a key to remove this window and enter the editor.

The menu window

At the top of the screen, there's a menu window containing a list of the currently available commands. This forms the gateway to all AMO S3 Basic's powerful editing features,, Command can be quickly executed by moving the mouse pointer over an item, and hitting the left mouse button. Each command is also assigned to a particular function key.

In addition to the main menu, there &ns also a number of other menus. The most important of these menus is the SYSTEM menu. This can be brought into view by either holding down the right mouse button, ex-pressing the shift key from the keyboard.

The SYSTEM manu contains a range of options such as LOAD, SAVE, NEW, etc. Like the main menu, all options can be executed using either the left mouse button, or by pressing an appropriate function key.

The information line

```
1   L=l   Oi    Text=40000     Chip=9i000     Fast=0     Editsexample
```

The markers at the far left display the editor mode ((I)nsert or (O)verwrite). There's also an indication of the (L)ine and (C)oiumn you *Are* presently editing. Alongside these markers is a list of three numbers;

TEXTs Measures the amount of memory which has been assigned to the editor window. This can be adjusted within All OS Basic using a simple SET BUFFER command from the SEARCH MENU.

CHIP; Free Chipmem

rASTs free Fistmem; Ulho,,eve,- poseibln, this **will be used**.

EDIT; Displays the name of the program you are currently editing.

Initially this area will totally blank, but when you load or save a Program to disc, the new filename will be automatically entered to the information line.

The editor window

The editor window forms the heart of the AMOS system, and allows you to type your Basic program listings directly from the keyboard. All text is inserted at the current cursor position,, which is indicated by a flashing horizontal line.

At the start of your session, the cursor will always be placed at the top left hand corner of the editing window. It can be moved around the current line using the left and right cursor keys.

Your line can be edited on a character by character basis using the Delete and Backspace keys., Delete erases the character immediately underneath the cursor, whereas Backspace deletes the character to the left of this cursor. As an example, type the lines

```
   print "AMOS"
```

When you press Return, your new line will be entered into AMOS Basic. Anything AMOS recognices as a command will be immediately converted to special format. All Basic commands begin with a Capital letter and continue in lower case. So the previous line will be displayed ass

```
   Print "AMOS"
```

Similarly, all AMOS variables and procedures are displayed in CAPITALS. This lets you quickly check whether you've made a mistake in one of your program lines,, Supposing for instance, you'd entered a line like:

```
   inpit "What's your name;";name$
```

This would be displayed ass

```
   Inpit "What's your name;";NAME*
```

Since INPIT is in UPPER case, it's immediately obvious that you've made an error of some sort,,

Ok- Now for a little fun. Move the cursor under the Print command you entered a few moments ago and type in the following lines of Basic Instructions.

```
   centre "<Touch 'n' Type Demo>
   do
     x*=inkey* : if x* <> "" then print x*
   loop
```

Don't forget to press the Return key after each and every line,, Wow move the cursor through your new program using the arrow keys,, Finally,, press the Fl to run this program.

The EDITOR WINDOW will disappear and a separate PROGRAM display will flip into place. The program now expects you to type in some text from the keyboard. As you can see, the program screen has its own independent cursor line,, This is totally separate to the one used by the editor. So you can play about as much as you like, without changing your current editing position„

After you've finished, press CTRL+C to abort the program. A thin line will now be displayed over the screen. This can moved using the up and down cursor arrows,

```
    Program Interrupted at line 4
  >»Loop
```

Pressing the space bar at this point would return you back to editor. But since we've already seen the editor, let's have a brief look at the Direct mode instead. Hit the ESCape key to flip this mode into place.

An introduction to Direct mode

DIRECT mode provides you with an easy way of testing your Basic programs. For the time being, we'll examine just a couple? of its more interesting features.

All direct mode commands are entered into a special screen which is completely inde pen tent, from the program display. You can move this screen up or down using the arrow keys.

At the top of the window, there's a list of 20 function key-assignments. These represent a list of commands which have been previously assigned to the various function keys. They can be accessed by hitting the left or right Amiga--keys in combination with one of the various function keys,

Whilst you're in direct mode, you can execute any Basic: instructions you like. The only exceptions *are* things like loops or procedures. As with the editor, all commands should be entered into the computer by pressing the Return key, Here are some examples:

```
Print 42
ANSWERS. Print ANSWER*?
Curs Off
Close Workbench              (Saves around 40k but ABORTS multi-
                               tasking operations!)
```

It's important to recognize that no matter what you do in direct mode, there will be absolutely no effect on the current program listing. So you can mess about to your heart's content, with no risk of deleting something in your Basic program,

It's now time to return to the Editor window, So wave a fond farewell to Direct mode, and enter the editor by pressing ESCape.

Loading a program

We'll now discuss the various procedures for loading and saving your programs on the disc. As usual, these options can be executed either from the MENU window or using a range of simple two-key commands from the editor. The fastest way to load a program is to hold down either of Amiga keys, and press the letter L.

You'll now be presented with the standard AMOS file selector window. Nowadays, file selectors have become a familiar part of most packages available on the Amiga. So if you've used one before, the AMOS system will hold no real surprises, However, since the file-selector is such an integral part of AMOS Basic, it's well worth explaining it in some detail.

The AMOS file selector

Selecting a file from the disc couldn't be easier. Simply move the cursor over the required filename so that it's highlighted in reversed text. To load this file into memory, click twice on the left mouse button. Alternatively,, you can enter the name straight from the keyboard, and just press Return,,

If you make a mistake, and wish to leave the selector without loading a file, move the mouse over the Quit button and select it with the left button!. AMOS will abort your operation and display a "Wot Done" message on the information line.

As an example, place you COPY of the AMOS program disc into the internal drive and press AMIGA+L to load a file. If you've been following out tutorial, AMOS will give you the option of saving the existing program first. Unless you've made any interesting changes,, press "N" to anter the file-selector. Otherwise, see "saving a program" for further instructions.

When the file selector appears, look out for a file with the name "Hithere,.AI1OS" „ Once you've found it, load it. The following listing will be loaded to amos basic™

```
        Rem Hi there AHOS user;
        Cls 0 : Rem Clear the sscreen with colour zero
        Do
          Rem get some random numbers
          X=Rnd(320)sY=Rnd(200]isT=Rnd(15):P=Rnd(15)
          Ink I„P; Rem add a 3.ittle colour
          Text X,, Y,"Hi there1" s Rem graphic text
        Loop
```

Move the text cursor over the text "Hi There!" and insert you own message- Mow press Fi to run the program,, The program display will rapidly fill up with do?:ens of copies of your text,, Press CTRL+C to exit fram this routine.

Saving & Basic program

Return to the editor window, and type ALT+S to save your current program onto the disc. If you feel like a change, hold down the right mouse key and click on the "Save as" option from the SYSTEM menu with the left button„ Either way you'll jump straight back to the AIIOS file selector window,.

You should now enter the name of your new file straight from the keyboard. As you type, your letters will appear in a small window at the bottom of the selector. Like the editor, there's a cursor at the current typing position. This cursor can be moved around using all the normal editing keys Finally, press Return to save your prog to disc.

Scrolling through your files

If your disc is reasonably full, the standard selection window won't be able to list the entire contents of your disc at once. You can page through the listing using the scroll bar to the left of the selection window«

Changing the current drive

To the right of the file window, there's a list of drive names,, The precise contents of the window will naturally depend on the devices you've connected to your Amiga,, If you have several drives, you can switch between them by simply clicking on the appropriate name,, (he direH-ory of this drive wlil now be entered into the selection window,

Changing the directory

When you search through the directly listing, you'll discover several names'with an asterix character "*" in front of them. These are not •files at all. They *are* entire directories in their own right.

You can enter one of these folders by selecting them with the left mouse button. You may then choose your files directly from this folder. Note that only the files with the current extension ".AMOS" will be displayed.

Once you.' ve opened a directory ,, ycm c:an set it as the def auIt using the SETDIR button. The next time you enter the file selector or obtain a directory listing with DIR, your chosen folder will be entered automatically. Similarly,, you can move back to the previous directory by clicking on the PARENT button.

Setting the search path

Normally, AMOS will search for all filenames with the extension ".AMOS", If you want to laod a file with another extension such as .BAK, you can edit the search pattern directly. This can be acomplished in the following way.

Move the text cursor to the PATH window by pressing with the up arrow from the keyboard. Now type your new path and hist Return. A full description of the required syntax can be found in the section on the DIR command.

WARNING!; AMOS uses its own individual search patterns which are very different from the standard Amiga Dos System. If you're unsure, delete the entire line up to the current VOLUME or DRIVE name and hit Return. This will present you with a full list of ALL the files on the present disc.

Using the file selector

Interestingly enough, it's also possible to call this file--selector directly from your own programs. For a demonstration, enter DIRECT mode and type the following lines

```
    Print Fselff*,,*)
```

After you've chosen a file, the name you've selected will be printed straight onto screen! See FSEL$ for a detailed explanation of this command.

Editor tutorial

We'll now have a brief look at some of the more advanced editing features available from the AMOS editor. We'll start by loading an example program from the disc:,. Just for a challenge, we've placed this in a separate MANUAL folder on the AMOS program disc.

Insert your COPY of the program disc into your Amiga'

Scrolling through a listing

Alongside the main editor window are two "scroll bars". These allow you to page through your listing with the mouse.

Hove the mouse pointer over the Vertical bar and hold down the left button. Wow drag the bar down the screen. The editor window will scrolls moothly downwards through the listing. You can also scroll the program using the Arrow Icons at the top and bottom of this bar. Clicking on these icons moves the line exactly one place in the requireddirection.,

At the far bottom of the editor window., there's a horizonal scroll bar. This can be used to move the window left and right in exactly the same way.

If you prefer to use the keyboard for your editing, you'll be pleased to discover that there are dozens of equivalent keyboard options as well. For example;

CTRL+UP Arrow	shift the listing to the previous page.
CTRL.+DOWN Arrow	moves the listing to the next page

All the keyboard options obey the same basic principles. So once you've familiarised yourself with one command, the rest are easy. A full list of these commands can be found towards the end of this chapter.

Now we've looked at the program. It's time to actually change something. Search through the program listing until you find the line:

```
        ALERT[5O,,"Alert box","","Ok","Cancel",,1,2]
```

This calls a Basic procedure which displays a working alert box on the screen ., The f ormat of this procedure is:

```
        ALERTLY coord,Title 1*,Title 2*,Button 1$,Button 2$,Paper,,Ink]
```

Let's change this alert to something a little more exciting., Hove the cursor over the above statement, and edit the line with the cursor keys so that it look like so:

```
        ALERTL50,"Exterminate!",,"Securitate","Yep!","Yep!",1,3]
```

Execute the program by pressing Fi or selecting RUN from the main menu. You'll be given the unique option of stopping the lamest Amiga-group in the World in its tracks. Select a button with the mouse and make your choicei

In practive, you can change the title and the buttons to literally anything you like. Feel free to use this routine in your own progs.

Hopefully, the above example will have provided you with a real spur to use procedures in your own programs,. In order to aid you. in this task, we've built a powerful range of special editing features into the AMOS editor.

Label/procedure searches 17

If your program is very long, it can be quite hard to find the starting points of your various procedure definitions. We've therefore included the ability to jump straight to the next procedure definition in your program, using just two keys (Alt+Arrow)

For an example, place the cursor at the start of the listing and , press Alt+down arrow. Your cursor will be immediately moved to the beginning of the first procedure definition in the current program (ALERT). You can repeat this process to jump to each procedure definition in turn,.

This system is not just limited to procedures of course.. It also works equally well with Labels or line numbers. So even if you don't needprocedures,, you'11sti11findauseforthisfeature.

Folding a procedure definition . . . 18

If you build up your programs out of a list of frequently used procedures, your lisings an easily be cluttered with the definitions of all your various library routines.

Fortunately, help is at hand. With a simple call to the Fold command, you can hide away any of your procedure definitions from your listings. These routines can be used in your program as normal, but their definitions will be replaced by a single Procedures statement. Example!;

Position the cursor anywhere in the definition of ALERT and click on the Fold/Unfold option from the menu window,, Bingi The contents of your procedure will vanish into thin air! Despite this, you can run the program with no ill effects. The only change has been in the appearance of the listing in the editor window.

Just select Fold/Unfold again, and your procedure will be expanded to it's fully glory.

It's also possible to fold ALL the procedures in your program at once. This uses an option on the SEARCH menu called "Close All". To bring the Sea r c h menu on to t he s creen ,, c 1 i c: k on the bu 11 on wi t h t he same name,, or press F5. from the keyboard. Wow select the Close All button to remove the procedure definitions from the current program.

The effect on EXAMPLE 3.1 is dramatic! The entire program now fits into just a single screen. So you can instantly see the procedures we've been using in the program. Each procedure definition can be edited individually by expanding it with the Fold/Unfold button. Or you can unfold the whole program with "Open All" from the Search menu.

Search/Replace

The search/replace commands provided by the AMOS Basic editor are accessed through a special Search menu which can be called up either from the menu window or by pressing function key F4.

Finding an item

We will continue our tutorial with a brief look at of some of the Search/replace instructions. Let's start with the FIND command.

This can be executed either directly from the Search menu or using the keys CTRL+F. When you select this command, you'll be asked to enter the search string.

For example, hit CTRL+F and type "Rem" at the prompt, AMOS ill now search for the next "Rem" statement in your program, starting from the current cursor position. If the search is succesful, then cursor will be replaced over the requested item.

The search can now be repeated from this point with the "Find Next" option (CTRL+W).

Replace

Supposing we wanted to change all the Rem statements in a program with the equivalent "'" characters. This could be accomplished with the "Replace" command.

In order to use this option,, it's necessary to define the replacement string. So the first time you call up replace, you will always be asked to enter this string from the keyboard.

Press CTRL+R, type in ' (apostophe) at the prompt and hit the return key to enter it into the computer. You now set the search string with the "Find" option like so:

* Press CTRL+F to select the FIND option.
* Type "Rem" into the information line.
t The cursor will then be moved straight to the next Rein statement in yourprogram1isting.

To change this to the replacement string and jump to the next occurrence, select Replace (CTRL+R) once again. Alternatively, if the Rem is in the middle of the line, you'll need to skip it, because AMOS only allows you to substitute a quote for this command at the start of a line. You can avoid this problem and jump directly to the next item in your program using "Find Next",

Cut and paste

The AMOS Block commands allow you to cut out parts of your programs and save them in memory for future use. Once you've created a block, you can copy it anywhere you. like in the current listing.

Here's an example of this feature in action. Let's take the previous ALERT program, and cut out a single procedure. Place the mouse pointer over the first line of the INVERT procedure, and depress the right mouse button. We can now enter this procedure into a block usinq the mouse. As you move the mouse, the selected *Area* will be highlighted in reverse.

We can now grab this area into memory using "Cut". When you press CTRL+C from the keyboard, the procedure will be removed from the listing and stored into memory. It's now possible to paste this block anywhere you like in your program. For the purposes of our example, move the text cursor down to the bottom of the listing, and call the Paste optin with CTRL+P. The INVERT procedure wlil now be copied to the current cursor position.

Multiple programs and accessories

Piuitiple programs

Although AMOS only allows you to edit a single program at a time,, there's no limit to the number of programs which can be installed into memory, other than the amount of available storage space- Once you've installed a program in this way, you can execute it straight from Editor window with the "Fain Other" option.

Supposing, for instance, you encounter a problem in one of your programs. AMOS will let you effortlessly swap your existing program into memory so that you can freely experiment with the various possibilities until you find a solution. After you've finished, you can now grab your new routine into memory with the cut option,, and flick back into your original program by pressing just two keys! The new routine can the be pasted into position, and you can continue with your program as before. The ability to stop everything and try out your ideas immediately, is incredibly valuable in practice.

Another possibility is to permanently keep all the most commondly neede utilities such as the sprite definer or the map editor in the memory. You can now access these utilities instantaneously., whenever you need them.

In fact, AMOS includes a special ACCESSORY system which makes this even easier. The utility programs can be given total access to all the memory banks in your main programs. So the sprite definer can grab the images straight from your current program,, and modify them directly,, This tehcnuque speeds up the overall development process by an amazing degree!

Let's have a quick demonstration of these facilities. Enter the following small prog into the editors

```
        Print "This is program One"
        Boom
```

We can now push this program into memory using the push command. This is called up by pressing AMIGA+P. You'll then be asked to enter the name of your program from the information line. Type in a name like "Programi" at this point. The edit screen will be cleared completely. The new window is totally separated from your original program. As a demonstration, enter a second routine like so:

```
        Print "This is program Two"
        Shoot
```

This program can now be executed from the editor window usinq RUN (F1). But when your return you can immediately jump to the old one with the "Flick" option. Try pressing AMIGA+F. As before,, you'll be asked to enter a name for your program,, Use a name like "prograrn2" for this purpose. The editor will now jump straight to your original program as if by magic It's possible to repeat this process to jump back and forth between the two programs. Each program is entirely independent and can have it's list of own banks and program screens.

So far,, we've only discussed how you can use two programs at a time. However, you can actually have as many program in memory as you like. These programs can be selected individually using the "Run Other" and "Edit Other" options from the Menu window,, When you call these commands, a special "program" selector will be displayed on the screen,, The p-nqram elector is almost identical to the familiar AMOS file selector/The only difference is that it allows you to choose a program from memory rather than from the disc, You cars select a program by simply highlighting it with the mouse cursor and clicking once on the left button.

Accessories

In order to distinguish accessories from normal Basic programs, they're assigned a ".ACC" extension instead of the more usual ".AMOS". Accessories can be loaded into memory like any normal program using the "Load Other" command.

Load Other presents you with a normal fileselector which can be used to load an accessory program from the disc. After the accessory has been installed into memory you will be returned straight back to your current program,. You can now run this accessory at any time using the Run Other option from the menu window. Simply move the mouse pointer- over your required accessory and press the left button™

Alternatively, you can load all the accessories from the current disc using the Accnes/Load feature. This option can be found on the System menu which is displayed when you hold down the right mouse button. Accnew/Load erases all existing accessories and loads a new set from the current disc.

For a demonstration, place the AMOS Program disc into your drive, and click on the Accnew/Load button fram the System menu.

The HELF' accessory will be quickly loaded into memory. HELP is a special accessory because it can be called up directly by pressing the H L E P key. We've packed this program with a11 the information you ' 11 need about the accessor yprograms supplied with AMOS Basic, All you need to do, is just follow the prompts which will be displayed on the screen.

Direct mode

The Direct mode window can be entered from the editor by pressing the ESCape key at any time. As a default,, the window is displayed in the lower half of the screen, with the program screen in the background,,

If you run a program that changes the screen format,, displays windows,, animates sprites etc, then all this screen data will remain intact. So you can move the DIRECT window around or flip back to the editor to make program changes without destroying the current program screen. This DIRECT mode window is totally independent and is displayed onitsownfront1eve1screen„

Whilst you're within direct mode you can type any line of AI10S Basic you wish.. The only commands you cannot use *Are* loops and branch instructions. You only have access to normal variables (as distinct from the loca bariatiles defined in a procedure).

Direct mode editor keys

ESCape	Jump to the editor window
Return	Executes the current line of commands
DELete	Delete character under cursor/
Backspage	Delete character to the left of the cursor
Left Arrow •	Move cursor left

Right Arrow	Hove cursor right
Shift+Left	Skip a word to the left
Shift+Right	Skip a word to the right
Shift DELete	Deletes entire line.
Shift BACK	Ditto
Help	Displays the function key definitions to the direct window.
Fl to FiO	These keys remember the last 10 lines you've entered from the direct mode. Fl displays the

latest one entered.,. F2 the second to last, etc, The memory area used by this system is always cleared when you return to the editor window or run one of your programs.

The menu window

There's a detailed explanation of all the options which are available from the main menu window-

Default menu

This gives you various commands that allow you to operate the editor, plus give you access to the block and search menus.

RUN	(Fl)	Runs the current program in memory
TEST	(F2)	Cheks the program syntax
INDENT	(F3)	Takes the current program andindentsthe listing,
BLOCKS MENU	(F4)	Displays the Blocks menu.
SEARCH MENU	(F5)	Displays the Search menu
RUN OTHER-	(F6)	Runs a program or accessory in memory
EDIT OTHER	(F7)	Edits a program which has previously installed into memory using the "Load Other" or "Accnew/Load"„
OVERWRITE	(F8)	Toggles between insert and overwrite -editing modes.
FOLD/UNFOLD	(F9)	Takes a procedure definition and folds it away inside yourprogramlisting„

Normally, it's possible to re--open a folded procedure by repeating the process. Place the cursor over a folded procedure and click on FOLD/UNFOLD. If you feel the need for extra security you can also call up a special LOCK accessory from the AMOS Program disc, This will ask for a code word, and will lock your procedures so that they can't be subsequen1lyexaminec!fromAII0SBasic,Simp1yfo1dyourrequired proceduers and load FOLD.ACC using the LOAD OTHERS command,, Full instructions are included with the utility.

The real beauty of this system is that it allows you to create whole libraries of your routines on the disc, These can be loaded into memory as a separate program (See LOAD OTHER). You can now cut out the routine you need and copy them directly into your main program. So once you've written a routine, cm can place it into a procedure and reuse it again and again.

1fyou'reintending to use this sytem, there are several points to concider.

* Whenever you fold or unfold a procedure a syntax check is made of the entire program,, If an error occurs the operatoni will not be performed. So it's vital that you keep back-up copies of all your procedures in Unfo

The system menu

LOAD	(SFT+F1 / AMIGA+L)	Loads an AMOS Basic Program
SAVE	(SFIi-F2 / AMIGA+S)	Saves the current Basic: Program
SAVE AS	(SFT+F3 /SFT+AM+S)	Saves the prog with another name
HERGE	(SHIFT+F4)	Enters the chosen prog at the current csrs position without erasing the current program.
MERGE ASCII	(SHIFT+F5)	Merges an Ascii version of an AMOS Basic program with the existing program in memory 25
AC.NEW/LOAD	(SHIFT+F6)	Enters a new accessory set from the disc
LOAD OTHERS	(SHIFT+-F7)	Loads a single accessory from the disc
MEW OTHERS	(SHIFT+F8)	Erases accessorie(s) from memory
MEW	(SHIFT+F9)	Erases the current program from memory
QUIT	(SHIFT+F10)	Exits AMOS and returns control to the CLI

The blocks menu

BLOCK START	(CTRL + B/Fl)	Sets the starting point for the current block
BLOCK END	(CTRL + E/F6)	Defines the end of a block
BLOCK CUT	(CTRL + C/F2)	Removes the selected block into memory 26
BLOCK PASTE	(CTRL + P/F7)	Pastes the block to the current csrs position
BLOCK MOVE	(CTRL + M/F3)	Move the block to the current cursor position erasing the original version completely
BLOCK STORE	(CTRL + S/F8)	Copies the block into memory.
BLOCK HIDE	(CTRL + H/F4)	Deselectstheblockyou'vehighIiqhted
BLOCK SAVE	(CTRL + F9)	Saves the current block on the disc as an AMOS program
SAVE ASCII	(CTRL + F5)	Stores your selected block on the disc: as a normal text file.,
BLOCK PRINT	(CTRL + Flû)	Outputs the selected block to the printer

Thesearchmenu

FIND	(ALT + Fl)	Enters a string of up to 32 chars and searches through your text until a match is found.
FIND NEXT	(ALT + F2)	Searches for the next match you specified
FIND TOP	(ALT + F3)	Searches from the top of program the string rather than starting from the crsr position
REPLACE	(ALT + F*i)	Activates REPLACE mode. The effect of this commandvariesdependingwhenit'sused:

* Before a FIND

You'll now be asked to enter the replacement string from the keyboard

* After a FIND

If the search operation was succesful, the text and the current cursor position will be swapped with the replacement string. REPLACE will now jump to the next occurrence of the search string.

REPLACE ALL.	(ALT + F5)	Replaces ALL copies of a word in your prog.
LOW <> UP	(ALT + F6)	Changes the case sensitivity used in search commands
OPEN ALL	(ALT + F7)	Opens all closed procedures in your program
CLOSE ALL.	(ALT + F8)	CLoses all procedures in your program
SET TEXT B	(ALT + F9)	SET TEXT BUFFER. Changes the » of chars available to hold your listings.
SET TAB	(ALT + F10)	Sets the number of chars which the crsr will be moved when the user presses the TAB key,,

Keyboard macros 28

= KEY*~- (define a keyboard macro)

KEY* (n)^ command t> '
comffiand$)-KEY$(n)

KEY* assigns the contents of command* to function key number n. (1-20) Keys from one to ten are accessed by pressing the function key in conjuction with the left Amiga button. Similarly, numbers from eleven onwards *&re* called with a right Amiga Fn combination.

Command* can be any string of text you wish., up to maximum of 20 characters. There *Are* two special characters which are directly interpreted by this functions

'	(Alt+Quote)	Generates a Return code
"	(single Quote)	Encloses a comment. This is only displayed in your key lists,. It's totally ignored by the macro routine. Examples:

```
? Key*(I)
Key*(2)--"Default"
Alt+F2

Key*(3)-"'Comment print"
```

In practice, this macro system can prove incredibly useful,, Klot only can you speed up the process of entering you Basic programs, but you can also define a list of standard inputs for your Basic programs. These would be extremely effective in an adventure game., as can be seen front the program EXAMPLE 3.2 in the MANUAL folder.

If you wish to generate a keypress which has no ASCII equivalent such as up arrow,, you can optionally include a scancode in these macros. This is achieved using the SCAN* function,,

=SCAN* (return a scan code for use with KEY*)

x$--Scan$(n, L",m])

n is the scancode of a key to be used in one of your macro definitions, m is an optional mask which sets the special keys such as CTRL, or Alt. in the following format:

Bit	Key Tested	Motes
0	Left SHIFT	
1	Right SHIFT	
2	Caps Lock	Either ON or OFF
3	CTRL.	
4	Left ALT	
5	Right ALT	
6	Left AMIGA	Commodore key on some keyboards
7	Right AMIGA	

If a bit is set to a one, then the associated button is depressed in your macro. Examples;

```
KEY*(4)="Wheeei"+Scan$($4C)
KEY$(5)="Page Up!"+Scan*(*4C,S00010000)
```

Conserving memory

CLOSE WORKBENCH (closes the workbench)

CLOSE WORKBENCH

Closes the workbench screen saving around 40K of memory for your programs! Example:

```
        Print Chip Free,Fast Free
        Close Workbench
        Print Chip Free,Fast Free
```

CLOSE WORKBENCH can be executed either from direct mode, or inside on of your Basic, programs, A Typical program line might be:

```
        If Fast Free=0 Then Close Workbench
```

This would check for a memory expansion and close the Workbench if extra memory was not available-

CLOSE EDITOR (close editor window)

CLOSE EDITOR

Closes the Editor window while your program is running, saving you more than 28K of memory. Furthermore, there's absolutely HO effect on your program listings!

If there's not enough memory to reopen the window after your program has finished, AMOS will simply erase your current display and revert back to the standard DEFAULT screen. You'll now be able to effortlessly jump back to the Editor with the ESCape key as normal,, What a terrific little instructioni

Inside accessories

We'll now explore the general techniques required to write your own accessory programs. These &r& really just specialised form of the multiple programs we discussed a little earlier. As you would expect, they can incorporate all the standard Basic instructions.

Accessories are displayed directly over your current program screen and the music, sprite, or bob animations i\r& automatically removed from the screen.

Your accessory should therefore check the dimensions and type of this screen using the SCREEN HEIGHT., SCREEN WIDTH and SCREEN COLOUR commands during its initialisation phase?,. If the current screen isn't acceptable, you may be forced to open a new screen for the accessory window or to erase the existing screens altogether with a DEFAULT instruction»

Any memory banks used by your accessory are totally independent of the main program. If it's necessary to change the banks from the current program, you can call a special BGRAB command,.

BGRAB (grabs the banks used by the current program)

BGRAB b

BGRAB "borrows" a bank from the current program and copies it into the same bank in your accessory. If this accessory bank already exists, it will be totally erased,, When the accessory returns to the editor, the bank you have grabbed will be automatically returned to your main program along with any changes, b is the number of a bank from 1 to 16.

Note that this instruction can only be used inside an accessory. If you try to include it in normal program, you'll get an appropriate arror message.

PRUN (run a program from memory)

PRUN "name"

Executes a Basic program which has been previously installed in the Amiga's memory. This commane! can be used either from the direct rnode, or within a program! In effect, PRUN is very similar to a standard procedure call, except that any bobs, sprites or music will be totally suspended.

Note that it's impossible to call the same program twice in the same session. After you've called it once, any further attempts will ignored completely.

~ PR6 FIRST* (read the first program loaded into memory)

p*==PRG FIRST*

This returns the name of the first Basic: program installed in the Amiga's memory,, It's used in conjunction with the PRG NEXT* command to create a full list of all the currently available programs.

=PRG NEXT* (returns the next program installed in memory)

p*=PRG NEXT*

PRG NETX* is used after a PRG FIRST* command to page through all the programs installed in Amiga's memory,, When the end of the list is reached., a value of "" will be returned by this function,, Example;

```
N*=Prg First*
While N*<>""
  Print "Program" "nM*
  N*=Prg Next*
Wend
```

=PSEL* (call program selector)

n*=PSELt("filter"[default*,titleIt,title2*]

PSFL* calls up a program selector which is indential to the one used by the "Run Other, Edit Other, Load Others, and New Others commands.. This can be used to select a program in the usual way. The name of this program will be returned in n*. If the user has aborted from the selector, n* will be set to an emptry string "".

"filter" sets the type of programs which will be listed by instruction. Typical values &re°.

"*.ACC"	List all the accessories in memory
"#.AMOS"	Only displays the AMOS programs which have been installed«
"t.t"	List all programs currently in memory.

For further details of the system see the MR command.

default*	holds the name of a program which will be used as a default.
titlei$,title*	Contains up to two lines of text which will be displayed at the top of the selector.

See EXAMPLE 3.4 in the MANUAL folder for a demonstration.

The HELP accessory 32

Whenever the HELP key is pressed from the Editor window, AMOS automatically executes an accessory with the name HELP.ACC if it's available. Unlike normal accessories, this is displayed directly over the editor window. Special access is provided to the current word you are editing. The address of this word is placed in an address register and can be rea d u s i ng the AREG func t ion.

The editor control keys

F i n a 11 y, he r e ' s a f u 11 1 i s t of t he va r i ous con t r o 1 keys an d ef f e c ts s

Special keys 33

ESC	Takes you to direct mode

Editing keys

Backspace	Deletes the character to the immediate left of crsr,
DELete	Deletes the character underneath the cursor
RETURN	Tokenises the current line. If you move onto a line and press RETURN it will split the line
SFT+BCKS/CTRL+Y	Deletes current line
CTRL+U	Undo. Return the last line when in overwrite mode.,
CTRL+Q	Erase the rest of chars in the line from crsr position
CTRL+1	Insert a line at the current position

The cursor arrows

Left,Right	Moves cursor one space to the left/right
Up,Down	Moves cursor one line up/down
SHIFT+Left,Right	Positionsthecursoroverthepreviou s/nextword
SHIFT+up,down	Move cursor to the top/bottom line of the current page
CTRL+up,down	Displays the previous/next page of program
SHIFT+CTRL+up.,dn	Move to start/end of text
AMIGA+up	Scrolls text up without moving the cursor
AMIGA+down	Scrolls text down under the cursor
AMIGA+1eft,right	Scroll program to the left/right on the current line

Program control

AM1GA+S	Saves your program under a new name
AMIGA+SHIFT+S	" " current name
AMIGA+L	Loads a program
AMIGA+P	Pushes the current program into a mem and creates a new program.
AMIGA+F	Flips between two progs stored in memory
AMIGA+T	Displays next program in memory.

Cut and Paste

CTRL+B	Set the beginning of a block
CTRL+E	Set end point of a block
CTRL+C	Cut block
CTRL+M	Block move
CTRL+S	Saves the block in memory without erasing it first
CTRL+P	Paste block at current cursor position
CTRL+H	Hide block.

Marks

CTRL+SHIFT+C0-9)	Defines a marker at the present cursor position,
CTRL+(0--9)	Jumps to a mark

Search/Replace

ALT+UP Arrow	Searches backwards through your program to the next 1inewhichcontainsalabe1orproceduredefinition»
ALT+DOWN Arrow	Searches down through *yur* program to find the next label or procedure definition
CTRL+F	Find
CRTL+M	Find Next
CTRL+R	Replace

Tabs

TAB	Move the entire line at the cursor to the next TAB pos,
SHIFT+TAB	Move the line to the previous Tab position
CTRL+TAB	Sets the TAB value

This chapter discusses the ground rules used to construct AMOS Basic programs and shows you how to improve your programming style with the help of AMOS Basic procedures.

Variables

Variables are the names used to refer to storage locations inside a computer. These locations hold the results of the calculations performed in one of your programs.

The choise of variable names is entirely up to you, and can include any string of letters or numbers. There *Are* only a couple of restrictions. All variable names MUST begin with a letter and cannot commence with an existing AMOS Basic instruction. However it is perfectly permissible to use these keywords inside a name. So variables such as VPRINT or SCORE are fine.

Variable names must be continuous, and may not contain embedded spaces. If a space is required,, it's a possible to substitute a "_" charac tsr ins tead.

Here are some examples of illegal names. The illegal bits are underlined to make things clearer.

```
        WHILE*, 5C, MODERN**, TOAD
```

Types of variables

AMOS Basic allows you to use three different types of variables in your programs.

Integers

Unlike most other Basics, AMOS initially assumes that all variables *Are* integers,. Integers *Are* whole numbers such as 1,3 or 8, and *&re* ideal for holding the values used in your games.

Since integer arithmetic is much faster than the normal floating point operations, using integers in you programs can lead to dramatic improvements in speed. Each integer is stored in four bytes and can range from --147'483'648 to +147'483'648. Examples of integer variables!

```
        A, NUMBER, SCORE, LIVES
```

Real numbers

In AMOS Basic these variables are always followed by a hash (•*) character. Real numbers can hold fractional values such as 3.1 or 1.5. They correspond directly to the standard variables used in most other versions of Basic. Each real variable is stored in four bytes and can range between **1E-14** and **1E-15**. All values are accurate to a precision of seven decimal diqits. Examples 5

```
        P**, NUMBER!*, TESTS
```

String variables

String variables contain text rather than numbers™ They are
distinguished from normal variables by the $ character at the end. The
length'of your text can be anything from 0 to 65'500 characters.
Examples of string variables;

```
        NAME*,, PATH*, ALIEN*
```

Giving a variable a value

Assigning a value to a variable is easy, Simply choose an appropriate
name and assign it to value using the "•=" statement,,

```
        VAR=10
```

This loads the variable VAR with a value of 10.

```
        A*="Hello"          ;
```

This assigns string "Hello" to a variable A*.

Arrays

Any list of variables can be combined together in the form of an array,
Arrays *are* created using the DIM instruction.

DIM (dimension an array)

DIM *yar(* x ,y, z ,»,.„)

DIM defines a table of variables in your AMOS Basic program,, These
tables may have as manu dimensions as you want, but each dimension is
limited to a maximum of 65'000 elements,, Examples

```
        Dim A$(10),B(10,,J,0),,C#(10,,10,I10)
```

In order to access an element in the array you simply type the array
name followed by the index numbers,, These numbers are separated by
commas and *a.r&* enclosed between round brackets ()..Note that the
element numbers of these arrays always start from zero. Examples

```
        Dim ARRAY(10)
        ARRAY(0)--i0:ARRAY(.1. )-™15
        Print ARRAY(1)3 ARRAY(0)
( result; 15 10 )
```

Constants

Constants *are* simply numbers or strings which are assigned to a
variable or used in one of your calculations™ They *Are* called constants
because they don't charge during the course of your program. The
•following values are all constants!

```
        1, 42, 3.141, "Hello"
```

As a default, all numeric constants are treated as integers., Any floating point assignments to an integer variable are automatically converted to a whole number before use. Examplesi

```
        A=3.141:Print A
( result; 3)
        Print 19/2
( result;; 9)
```

Constants can also be input using binary or hexadecimal notation. Binary numbers *Are* signified by preceding them with a *: character, and hexadecimal numbers *are* denoted by a $ sign,. Here's number *2b5z*

```
        Decimals        255
        Hexadecimal:    *FF
        Binary:;        $11111:1.11
```

Mote that any numbers you type in AMOS Basic are automatically converted to special internal format. When you list your program these numbers are expanded back into their original form. Since AMOS Basic prints all numbers in a standard way, this will often lead to minor discrepancies between the number you entered and the number which is displayed in your 1 i s t i ng . However t he va 1 ue of t he n umber wi. 11 rema in exactly the same. Floating point constants are distinguished from integersbyadeci*tn*a1po1nt.Ifthispointisnotused,,thenumberwill always be assumed to be an integer, even if this number occurs inside a floating point expression. Take the following examples

```
        For X=l To 10000
          A#=A#+2
        Next X
```

Every time the expression in this program is evaluated,. the "2" will be laboriously converted into a real number. So this routine will be inherently slower than the equivalent program belows

```
        For X=l To 10000
          A#=A#+2.0
        Next X
```

This program executes over 252 faster than the original one because the constant is now stored directly in floating point format. You should always remember to place a decimal oint after a floating point constant even if it is a whole number. Incidentally, if you mix floating point numbers and integers, the result will always be returned as a real number. Examples

```
        Print 19.0/2
( results 9.5 )
        Print 3.141+10
( result; .1.3.141 )
```

Arithmetic operations

The following arithmetic operations can be used in a numeric expressions

```
        ^       power
        S * d i v i d e ^ n d m 11 t i p 1 y
        MOD     modulo operator (remainder of a division)
```

+ -	plus and minus
AMD	logical AND
Oft	logical OR
NOT	logical WOT

We've listed these operations in descending order of their priority. This priority refers to the sequence in which the various sections of an arithmetic expressions *are* evaluated.. Operations with the highest priority are always calculated first.

INC (add 1 to an integer variable)

INC var

INC adds 1 to an integer variable using a single 68000 instruction. It is logically equivalent to the expression var==var+l, but faster. Example:

```
        A=10sInc AsF'rint A
( results 11 )
```

DEC (subtract 1 from an integer variable)

DEC var

This instruction subtracts 1 from the integer variable \>*&r.* Example;

```
        A=2sDec AsF'rint A
( results 1 )
```

ADD (fast integer addition)

ADD v,exp [,base TO top]

The standard from of this instruction immediately adds the result of the expression exp to the integer variable v. It's equivalent to the line: V=V+EXP

The only significant difference between the two statements is that ADD performs around 40*; faster. Note that, the variable v must be an integer. Examples

```
        Tiiner=0
        For X=l To 1000
          Add T,X
        Next X
        Print T,Timer
( results 500500 7 )
```

The second version of ADD is a little more complicated. It is effectively identical to the following code (but faster);

```
        V=V+A
        If V<Base Then V=Top
```

Example;

```
        Dim A(10)
        For X=0 To 10;A(X)»X;Next X
        V=0
        Repeat
          Add V,l,l To 10
            Print A(V)
        Until V=100:rGm This is an infinite loop as V is always less
                            than 10i
```

As you can see. ADD is ideal for handing circular or repetitive loops in your games.

String operations 40

Like most versions of Basic, AMOS will happily allow you to add two strings together,.

```
        A*="AH0S"+" Basic"
        Print A$
( results AMOS Basic )
```

But AMOS also lets you perform subtraction as well. This operation works by removing all occurrences of the second string from the first,

```
        Print "AMOS BASIC"-"AM0"
( result; S BASIC )
```

Comparisons between two strings are performed on a character by character basis using the Ascii values of the appropriate letters:;

```
        "AA"<"BB"
        "Filename"3*"Filename"
        "X&">"XH"
        "HELLO"<"hello"
```

Parameters 41

The values you enter into an AMOS Basic instruction are known as parameters, i.e

```
        Inc N
        Add A.,10
        Ink 1,2,,3
```

The parameters in the above instructions are N,A,10,1,2 and 3 respectively. Occasionally, some of the parameters of a command can be ommitted from an instruction. In this case., any unused values will autoinatically be assigned a number *by* default., Examples

```
        Ink 5,,
```

This changes the ink colour without affecting either the paper or outline colours.

Line numbers and labels

Labels

Label* arp just a convenient way of marking a point in your AMUb Basic nroqrams. They consist' of a string of characters formed using the same rules as AMOS variables. Labels should always be placed at the start of the line, and must be followed immediately by a »:" character., There should be no spaces between the label and the colon., Examples

```
        TESTLABEL:
        Print "Hi There!"
        Goto TESTLABEL
```

This program can be aborted by pressing CTRL+C

42

Procedures

Procedures allow you to concentrate your efforts on just one problem at a time without the distractions provided by the rest of your program. Once you've written your procedures you can then quickly combine them in your finished program, AMOS procedures are totally independent program modules which can have their own program lines,, variables,, and even data statements.

PROCEDURE (create an All OS Basic procedure)

```
Procedure MA PI EC parameter list]
:     :
End Proc[Expression3
```

This defines an AMOS Basic procedure called NAME. NAME is a string of characters which identify the procedure., It is constructed in exactly the same way as a normal Basic variable. Note that it's perfectly acceptable t ouse identical names for procedures, variables and labels. AMOS will automatically work out which object you are referring to from the context of the liDe-

procedures are similar to the GOSUB commands found in earlier versions of Basic™ Here's an example of a simple AMOS procedure;

```
        Procedure ANSWER
          Print "Forty-Two!"
        End Proc
```

See how the procedure has been terminated with an END PROC statement. You should also note that the Procedure and the End Proc directives &r& both placed on their own separate lines. This is compulsory,,

If you type the previous procedure into AMOS Basic as it stands, and a1tempt to run i t, n othin g wi11 hap pen„ That's be cause you haven't actually called the new procedure from your Basic Program,, This can be achieved by simply entering its name at the appropriate point in the program. As an example, enter the following line at the start of the program and run it to see the result of the procedure,,

```
        ANSWER
```

IMPORTANT! When you are using several procedures on the same line, it's Advisable [illegible] avoid the risk of the procedure being confused with a label. For examples

```
TEST s TEST : TEST      Performs the test three times.
TEST:TEST:TEST          Defines Label TEST and executes test 2x
```

Alternatively, you can preclude your Procedure calls with a Proc statement like so:

```
Proc ANSWER
```

Example:

```
Proc ANSWER
Procedure ANSWER
  Print "Forty-Two"
End Proc
```

If you run this program again, the procedure will be entered,, and the answer will be printed out on the screen. A11hough the procedure definition is positioned at the end of the program, it's possible to place it absolutely anywhere,, Whenever AMOS encouters a Procedure statement, it installs the procedure and immediately jumps to the final End Proc. This means there is no danger of accidentally executing your procedure by mistake. Once you've created a procedure,, and tested it to your satisfaction, you can suppress it in your listings using the fold option from the main menu.

Thesefo1dingprocedures r-educe the apparent complexity of your listings and allow you to debug large programs without the distractions of unimportant details. You can restore your procedure listings to the screen at any time by selecting the 'unfold menu option'.

Local and g1oba1 variab1es

All the variables you define inside your procedures are independent of any other variables used in your program,. These variables *Are* said to be "local" to your particular procedure. Here's an example which illustrates this::

```
A=1000:B=42
TEST
Print A,,B
Procedure TEST
Print A,B
End Proc
```

It should be apparent that the names A and B refer to completely different variable depending on whether they *Are* used inside or outside the procedure TEST. The variables which occur outside a procedure are "global" and cannot be accessed from within it. Let's take an other-examples

```
Dim A(iOO)
For V=l To 100s A(V)=V:Wext V
TEST_FLAG=1
APRINT
End
Procedure APRINT
  If TE3T._FLAG=i
   For P=l To 100
     Print A(P)
   Next p
  Endif
```

```
        End Pros::
```

This program may look pretty harness but it contains two fatal errors.

Firstly, the value of TESTJ1.AG inside the procedure will always have a valup oiF zero. So the loop~~betwe.en the IF and the ENDIF will never be performed. That's because the version of TEST....FLAG within the procedure is completely separate from the copy defined in the main program. Like all variables, it's automatically assigned to zero the fist time it's used.,

Furthermore, the program won't even run! Since the global array a() has been defined outside ARPINT, AMOS Basic will immediately report an "array not dimensioned" error at the lines

```
        Print A(P)
```

This type of error is extremely easy tomake. So it's vital that you treat procedures as separate prog rams wi th thei r own independen t set of variables and instrcutions.,

There *are* a couple of extensions'to this system which make it easy for you to transfer information between a procedure and your main program. Once you're familiar with these commands you'll have few problems in using procedures successfully in your programs.

Parameters and procedures

One possibility is to include a list of "parameter definitions" in your procedure. This creates a group of local variables which can be loaded directly from the main program- Here's an examples

```
        Procedure HELLOCWAilE*]
          Print "Hello "jNAME*
        End Proc
```

The value to be loaded into NAME* is entered between square brackets as part of the procedure call. So the HELLO procedure could be performed in the following ways:

```
        Rem Loads K1$ into NAME* and enters procedure
      . Input "What's your name";n*
        HELLOCN*]
        HELLOC"Stephen"]
```

As you can see, the parameter system is general purpose and works equally well with either variables or constants,, Only the type of the variables *Are* significant.

This process can be used to transfer integer,, real or string variables. However you cannot pass entire *Arrays* with this function. If you want to enter several parameters you should separate your variables using commas. For examples

```
        Procedure POWER[A,B]
        Procudure MERGE[A*,B*.,C*3
```

These procedures might by called using lines like:

```
        POWER!!10,3]
        MERGEL"One","TWD","Three":i
```

Shared variables

Another way of passing data between a procedure and the main program is to use the SHARED instruction.

SHARED (defina a list of global variables)

SHARED variable list

SHAFTED is placed inside a procedure definition and takes a list of AMOS Basic variables separated by commas. These variables *Are* now treated as global variables., and can be accessed directly from the main program. Any arrays which you declare in this way should of course have been previously diiiiensioned in your main program. Examples

```
    A=1000:B=42
    TEST
    Print A?B
    Procudure Test
      Shared A,,B
      A=A+B:B=B+10
    End Proc
```

TEST can now read and write information to the global variables A and B. If you want to share an array you should define it like sos

```
    Shared A(),B$(),C*() s Rem Share arrays A,Bit and C*
```

GLOBAL (declare a list of global variables from the main program)

GLOBAL variable list

When you're writing a large prog rani,, it's commonplace for a number of procedures to share the same set of. global variables. This provides a simple method of transferring large amounts of information between your various procedures. In order to simplify this process, we've included a single command which can be used directly in your main program. GLOBAL defines a list variables which can be accessed anywhere inside your Basic program, without the need for an explicit SHARED statement in your procedure.

Returning values from a procedure

If a procedure needs to return a value which is only local to itself, it must use the following command so that it can inform the calling PROCEDURE command where to find the local variable

PARAH (return a parameter from a procedure)

PARAM

The PARAM functions provide you with a simple way of returning a result from a procedure. They take the result, of an optional expression in the END PROC statement, and return it in one of the variables PARAM, PAR Alitf, or PARAfl$ d spend ing on its type,, Examples

```
        MERGE_.STRIWGS["Afflos"," "..“Basic11:!
        Print PARAM*
        Procedure HERGE_STRINBS[At,B*,Ct]
          Print A*,B*,Ct
        End F'roc
```

Nntp that END PROC may only return a single parameter in this way. The PARA11 functions will always contain the result of the most recently executed procedure. Here's another example, this time showing the use of the PARAI18 function. . .. .

```
        CUBE[3,0]
        Print Param#
        Procedure CUBE[A$T|
          Ctt=CUBE8*CUBEtt*CUBEtt
        EndProc[Ctt]
```

Leaving a procedure

POP F'ROC (leave a procedure immediately)

POP PROC

Normally, procedures will only return to the main program when the END PROC instruction is reached. Sometimes., however,, you need to exit a procedure in a hurry. IN this case you can use the POP PROC function to exitimmediate1y.

Local DATA statements

Any data statements defined inside one of your procedures *are* held completely separately from those in the main program. This means each procedure can have its own individual data areas.

Hints and tips

Here are a few guidelines which will help you make the most out of your AMOS Basic procedures:

* It's perfectly legal for a proceduces to call itself, but this recursion is limited by the amount of space used to store the local variables. If your program runs out of memory you'll get an appropriateerror„

* All local variables are automatically discarded after the procedure has finished executing.

Memory banks

AMOS Basic includes a number of powerful facilities for manipulating sprites,, bobs and music. The data required by these functions needs to be> •stored along with the> Bi*i:i.c: prrtgram.. AMOS>`B«i*sic; u^e?is V* ?>p£?c;i.^it, s<?"t of 15 sections of memory for this purpose called "banks".

Each bank is referred to by a unique number ranging from 1 to 15. Many'of these banks can be used for all types of data, but some are dpdirAtPd solely to one sort of information such as sprite definitions. All sprite images are stored in bank 1. They can be loaded into memory using a line like:

```
        Load "AMOS.....DATAsSprites/Or.topus.abk"
```

There are two different forms of memory banks Permanent and temprorary. Permanent banks only need to be defined once, and *ans* subsequently saved along with your program automatically. Temporary banks are much more volatile and are reinitialized *every* time a program is run. Furthermore, unlike permanent banks,, temporary banks can be erased from memory using the CLEAR command.

Types of memory bank

AMOS Basic supports the following types of memory banks

Class	Stores	Restrictions	Type
Sprites	Sprite or bob definitions	Only bank 1	Permanent
Icons	Holds icon definitions-	Only bank 2	Permanent
Music	Contains sound track data	Only bank 3	Permanent
Amal	Used for AMAL data	Only bank 4	Permanent
Samples	The Sample Data	banks 1-15	Permanent
Menu	Stores MENU def in it i on	banks 1-15	Permanent
Chip work	Tempo*rury* workspace	banks 1-15	Temporary
Chip data	Permanent workspace	banks 1-15	Permanent
Fast work	Ternpora*ry* workspace	banks 1-15	Temporary
Fast data	Permanentworkspace	banks 1-15	Permanent

RESERVE (reserve a bank)

RESERVE AS type,bank,length

The banks used by your sprites or bobs are allocated automatically by AMOS. The RESERVE command allows you to create any other banks which you might require. Each different type of bank has its own unique version of the RESERVE instruction.

RESERVE AS WORK bankno, length

Reserves "length" bytes for use as a temporary workspace. Whenever possible this memory area will be allocated using fast memory, so you shoudn't call this command in conjunction with instructions which need to ace:ess to Ami*q*a 's b 1 i 11er chi p.

RESERCE AS CHIP WORK bankno,length

Allocates a workspace of size "length" using chip ram. You can check whether there's enough chip ram available with the CHIP FREE function.

RESERCE AS CHIP DATA bankno,length

Reserves "length" bytes of memory from chip ram. This bank will be automatically saved along with your All OS programs..

BnnK *may be* an;r number between 3. ond IS. Since banks i -to 5 «i-« normally reserved by the system,, it's wisest to leave them alone. Note that the only limit to the length of a bank is the amount of available memory.

LISTBANK (list the banks in use) .. .

LTSTBANK lists the numbers of the banks currently reserved by a program, along with their location and size. The listing is produced in the following formats

```
        Number     Type     Start     Length
```

Normally the length of a bank is returned in bytes, but in case of sprites and icons the value represents the total number of linages in the bank instead. The reason for this is that the storage of each image can be anywhere in the Amiga's memory, the bank is therefore not a continuous block of memory. So don't BSAVE a sprite bank,, simply use SAVE "filename.abk"

Deleting banks

During the course of a program you may need to clear some banks from the memory so as to load in additional data. Sprites may need to change for a new part of a game or a special piece of music is required to be played. The ERASE command gives you quick control for data deletion,,

ERASE (delete a bank)

ERASE b

ERASE deletes the contents of a memory bank. The bank number b can range from 1 to 15. Note that any memory used by this bank is subsequently freed for use by your program.

Bank parameter functions

If you want to have direct access to the bank data using commands such as poke, doke and loke then use these commands to find a bank's address in memory and its size.

=START (get the start address of a bank)

s=START(b)

This function returns the start address of bank uninber b. Once it's been removed, the location of the bank will never subsequently change, So the result of this function will remain fixed for the lifetime of t he ban k. Ex am pies

```
        Reserve As Work 3,2000
        Print Start(3)
```

^LENGTH (Get the length of a bank)

l=length(b)

The LENGTH function returns the length in bytes of bank number b. If

the bank contains sprites then the number of sprites or icons will be returned instead. A value of zero indicates that bank b does not exist.
Exaple:

```
        Reserve as work 6,1000
        Print Length(6)
        Erase 6
        Print Length(6)
```

Loading and saving banks - 51

Some programs will require many banks of information, a good example is an adventure. This would need to load various graphics and sounds for the different locations within the games domain,. An Amiga 500 would have great difficulty holding all this data at once and so it's best to simply load the data at the appropriate time of use.

LOAD (Load one or more banks)

LOAD "filename"[,n]

The effect of this command varies depending on the type of file you *a,r<s* loading. If the file holds several banks, then ALL current *memory* banks will be erased before the new banks are loaded from the disc. However if you're loading just a single bank,, only this bank will replaced. The optional destination point specifies the bank which is to be loaded with your data,, If it's omitted, then the data will be loaded into the bank from which it was originally saved.

Sprite banks are treated slightly differently. In this case the parameter n toggles between two separate loading modes. If n is omitted or is assigned a value of zero, the current bank will completely overwri11enbythenewsprites.,Anyothervaluefornforcesthenew sprites to be ^appended* to this bank. This allows you to combine several sprite files into the same program. Examples

```
        LOAD "Ai10S....DATA;Sprites/0ctopus.abk"
```

SAVE (E>ave one or more banks onto the disc)

SAVE "filenami"[,n] • • • ' . . .

The SAVE command saves your memory banks onto the disc, There *Are* two possib1eformatss

```
        SAVE "filename.ABK"
```

This saves *ALL* currently defined banks into a single file onto your disc.

```
        SAVE "filename.ABK",n
```

The expanded form just saves memory bank number n. One should also be sure to use the extension ABK at the end of the filename as this will ens;ure you can identify that the file contains one or more? memory banks.

BSAVE (Save an'unformatted block in binary format)

```
BSAVE file*, start TO end
```

The memory stored between "start" and "end" is saved on the disc in file*. This data is saved with no special formatting.. Examples

```
        BSAVE "Test",.Starts?) TO Start(7)+Length(7)
```

The above example saves the data in memory bank 7 to disc. The difference between this file and a saved file as a normal bank is that SAVE writes out a special blank header that contains information concerning the bank,, This header is not present with a BSAVED file so it cannot be loaded using LOAD.
WARNING; The sprites an icon banks are not stored as one chunk of memory. Each object can reside anywhere in memory. Because AMOS uses this flexible system of data storage you simply can't save the memory bank using BSAVE.

BLOAD (load binary information into a specified address or bank)

```
BLOAD file*, addr
```

The BLOAD command loads a file of binary data into memory., It does not alter the incoming information in any way,, There &re two forms of this function.

```
        BLOAD file*, addr
```

File* will be loaded from the disc into the address addr.

```
        BLOAD file*., bank
```

File* will be loaded into bank. This bank must have been previously reserved, otherwise an error will be generated. Also be sure not to load a file that is larger than the reserved bank, otherwise it will over run the bank and start corrputing other areas of memory!

Memory fragmentation

Sometimes, after a busy editing session, you may get an "Out of Memory" error, even though the information line implie

Finding space for your variables

A=. a default, all variables are stored in a memory Are A of exactly 8 k in length,, Although this may seem incredibly meagre, it's easily capable of holding around 2 pages of normal text, or 2000 numbers. We've intentionally set it as small as possible so as to maximize the amount of space available for your screens and memory banks.

SET BUFFER (set the size of the variable area)

```
SET BUFFER n
```

Sets the size of the variable area in your current program to "n" kilobytes. This must be the FIRST instruction in your program (excluding Rems). Otherwise you'll get an appropriate error message,, For an example of this feature see EXAMPLE 4..1 in the MANUAL folder.

SET BUFFER should be used in your program whenever you get an "out of string space error". Increase the value in 5k increments until the error disappears™ If you run out of memory durinq this pracess,, you'11 propably need to reduce the requirements of your program in some way. See the CLOSE WORKBENCH and CLOSE EDITOR commands for more details.

=FREE (return the amount of free mem,, in the variable area)

f=FREE

FREE returns the number of bytes which *Are* currently available to hold your variables. This value can be increased as required using the previous SET BUFFER command.

Whenever FREE is called, the variable area is reorganized to provide the maximum space for your variables. This process is known as "garbage collection", and is normally performed automatically.

Due to the power of AMOS Basic:,, the entire procedure is usually accomplished practically instantaneously. But if your variable area is bery large and you're using a lot of strings., the garbage collection routine might take several seconds to complete. Conceivably, this could lead to a unexpected delay in the execution of your programs. Since the garbage collection is totally essential, you may need to add an explicit call to the FREE command when it will cause the least amount of harm in your program.

=LEFT$= (return the leftmost characters of a string)

d*=LEFT*(s*,n)

This instruction works like in nearly any Basic language (for example, AmiqaBasic),, Examples

```
        B$="Hello! This is Ronnie!"
        U=Left*(B*,9)
        Print L *
( results Hello! Th )
```

=RIGHT$= (return the rightmost character of a string)

d*=RIGHT*(s*,n)

Same as the LEFT* -instruction,, but takes the rightmost characters.

```
        Print Right*("AMOS Basic".,5)
( result; Basic )
```

=HIM= (return a string of characters from within a string) 55

d*=MID*(s*,p,n)
MID$(d$,p,n)=s$

The MID* function returns the middle section of the string held in s$. p denotes the offset of characters to the start of this substring;, and n holds the number of characters to be fetched. If a value of "n" is not specified in the instruction then the characters will be read right up to the end of your string. Examples

```
        Print Hid*("AMOS Basic",A)
( result: Basic )
```

There is also a I1ID$ instructions

```
        MID$(d$,p,n)=s$
```

This version of HIM loads "n" characters into d$ starting from position p+i in s*. If a value of n is not specified directly then characters will be replaced up to the end of the source string s*. This kind of instruction is also possible when using !...EFT$ and RIGHT*. Here's an examples

```
        A$="AMOS ****#"
        Mid$(A$,5)="Magic"
        Print A$
( results AMOS Magic )
```

=IWSTR (search for occurrences of a 56

string within another string)

f=INSTR(d*,s$ up])

IMSTR allows you to search for all occurrences of one string inside another. It is often used in adventure games to split a complete line of text into its individual commands,, There are two possible formats of the INSTR function.

f=INSTR(d$,s$)

This searches for the first occurrence of s$ in d$. If the string is found then its position will be returned directly,, otherwise the result will be set to zero. Examples:

```
        Print Instr("Ai10S BASIC",,"AMOS")
( result: 1 )
        Print InstrC'AMOS BASIC","S")
( results 4 )
        Print Instr("AMOS BASIC","AMIGA")
( result; 0 )
```

```
        Do
          Input "String to be searched";D*
          Input "String to be found";S$
          X=Instr(D*,S$)
          If X=0 Then Print S$;" Wot found"
          If XOO Then Print S*;" Found at position '';X
        Loop
```

Normally the search will commence from the first character in your text string (dt). The secant version of INSTR lets you. test a specific section in the string at a time.

p is now the position of the beginning of your search. All characters &re numbered from the left to right starting from zero. Therefore p ranges from 0 to LEKi(st). Examples

```
        Print InstrC'AMOS BASIC", "S",5)
( result: 8)
```

=UPPER* (convert a string of text to upper case) 5?

s*=UPPER*(n*)

This function converts the string in n$ into upper case (capitals) and p1acestheresu11intos$„Examp1es

```
        Print Upperf("AmOs BaSic")
( results AMOS BASIC )
```

=LOWER$ (convert a string to lower case)

s*=L0UER*(n$)

LOWER* tr«nslat«« «i 1 t,-,» = h.»„• ~ .=t „ -«. L „ „* :!.„+.= ICWOK- ,~aeG_ This is especially useful in adventure games., as you can convert all the user's inpiitintoastandardformatwhichismucheasiertointerpret.

I Example:

```
        In put "Con tinue (Yes/No)";ANSWER*
        ANStoT£R*=Lower*( ANSWER*) 5 If ANSWER*="no" Then Edit
        Print "Continuing with your prog.,.."
```

=FLIP* (invert a string)

f*=FLIP*(n*)

FLIP* simply reverses the order of the characters held in n*.

=SPACE* (space out. a string)

s*=SPACE*(n)

Generates a string of n spaces and places them into s*. Examples

```
        Print "Twenty" ; Space*(20)5 "spaces"
```

=STRIN6* (create a string full of a*) 58

s*=STRING*(a$,n)

STRING* returns a string with n copies of the first character in ais

```
        Print String*("The cat sat on the mat", 1.0)
( results TTTTTTTTTT )
```

=CHR* (return Ascii character)

s*=CHR*(n)

Creates a string containing a single character with Ascii code n„

=ASC (get Ascii code of a character)

c==ASC(a*)

ASC supplies you with the internal Ascii code of the first character in the string a$s

```
        Print Asc("B")
( results 66 )
```

=LEN (returns the number of characters stored in a*)

This way you. can get the length of a strings

```
        Print Len("12345678")
( results 8 )
```

-=VAL (convert a string to number)

```
v=VAL(x$)
v#=VAL(x$)
```

VAL converts a list of decimal digits stored in x$ into a number. If this process fails for some reason, a value of zero will be returned instead,, Examples

```
        X=Val("1234):Print X
( results 1234 )
```

=STR* (convert a number to a string)

s*=STR*(n)

STR$ converts an integer variable into a string,, This can be *"jery* useful because some functions., such as CENTRE, do not allow you to enter numbers as a parameter. Example;

```
        Centre "Memory left is "+Str*(Chip Free)*" Bytes."
```

Do not confuse STR$ with STRING*.

Array options

SORT (sort all elements in an array)

```
SORT a(0)      ;
SORT a#(0)     ;  The SORT instruction arranges the contents of any
SORT a*(0)     !  array into ascending order. This array can contain
               ;  either strings,, integers,, or floating point numbers.
```

The a$(0) parameter specifies the starting point of your table. It must always be set to the first item in the array (item number 0). Example:

```
        Dim A(25)
        P=0
        Repeat
          Input "Input a number (0 to stop)";A(P)
          I n c P
        Until A(P-1)=O Or P>25
        Sort A(0)
        F o r 1 = 0 t o P - l
          Print A(I)
        Kiext
```

MATCH (search an *a.rt~&y)*

```
r=MATCH(t(O),s)
r=MATCH(t«(O),s«)      MATCH searches through a sorted array for the
r=MATCH(t*(0),s$)      value s. If this is succesfully found then r
                       will be negative., Taking the absolute value of
```

this figure? will provide you with the item which came closest to your original search parameter,,

Wote that only arrays with a single dimension can be checked in this way. You'll also need to sort the *arr&y* with SORT before calling this function« Examples

```
Read N
Dim Dt(N)
For 1=1 to N
  Read Dt(I)
Next I
Sort D$(0)

  Input A$
  If A$ = "L"
    For 1 = 1 to NsPrint D$(I):Next I
  Else
    P0S=!1atch(I)$(0),A$)
    If P0S>0 Then Print "Found",D$(PQS);" In Record ";POS
    If P0S<0 And Abs(POS)<=N Then Print A*,"Not Found. Closest
        ;                                 To %D$(Abs(P0S))
    If P0S<0 And Abs(POS)>N Then Print A$, "Not Found. Closest
                                                     To ";D$(N)
  Endif I
Loop
Data 10,"Adams",."Asimov","Shaw","Heinlien","ZeIazny","Foster"
Data "Niven","Harrison","Pratchet%"D:Lckson"
```

Note that MATCH could be used in conjunction with the 1KISTR function to provide a powerful parser routine. This might be used to interpret the instructions you entered in an adventure game.

AMOS Basic provides you with everything you need to generate some amazing graphics. There's a comprehensive set of commands for drawing rectangles, circles and polygons,, As you would expect from the Amiga, all operations *Are* performed practically instantaneously. But even here AMOS Basic has a trick or two up its sleeve.

The AMOS graphical functions work equally well in all the Amiga's graphics modes INCLUDING hold and modify mode (HAM). It's therefore possible to create breathtaking HAM pictures directly within AIIOS Basic!

Furthermore, you're not just limited to the visible screen. If you've created an extra large playing area, you'll be able to access *every* part of your display using the standard drawing routines. So it's easy to generate the scrolling backgrounds required by arcade games such as Defender.

Colours

The Amiga allows you to display up to 64 colours on the screen at a time. These colours can be selected using the INK,,COLOUR and PALETTE commands.

INK (set colour used by drawing operations)

INK col[, paper][,border]

"col" specifies the colour which is to be used for all subsequent drawing operations. The colour of every point on the screen is taken from one of 32 different colour registers. These registers can be individually set with a colour value chosen from a palette of 4096 colours.

Although the Amiga only provides you with 32 actual color registers,, AMOS lets you use colour numbers ranging from 0 to 63. This allows you to make full use of the colours available from the Half-Bright and HAH modes respectively. A detailed explanation of these modes can be found in the Screens chapter.

The "paper" colour sets the background colour fill patterns generated by the SET PATTERN command,,

The "border" colour selects an outline colour for your bars and polygons., This option can be activated using the SET PAINT command like sos

```
        Set pattern 0 : Set paint 1
        Repeat
          C=Rnd(i6):Ink 16-C,0,C
          X=Rnd(320)--20sY=Rnd(200)-20:S=Rnd(i00)+10
          Bar X,Y to X+S,Y+S
        Until Mouse Key
```

Simply include "empty" commas at the appropriate places in the instruction. For examples

```
        Ink ,,5 : Rem  Just sets the border colour
```

COLOUR (assign a colour to an index) 62

COLOUR index,tRGB

The COLOUR instruction allows to assign a colour to each of the Amiga's 32 colour registers,

"Index" is the number of the colour you wish to change,, and can range from 0-31. As you may know, any colour can be created by (nixing specific amounts of the primary colours Red, Green and Blue, The shade of your colour is completely determined by the relative intensities of the three components .

The expression $R6B consists of three digits from 0 to F. Each component sets the strength of one of the primary colours, Red (R), Green (G) or Blue (E«). The size of the components is directly proportional to the brightness of the associated colour„ So the higher values, the brighter the eventual colour.

Hex Digit	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Decimal	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

HAM and Extra Half-Bright modes use these indices siighty differently. See Chapter 9 for more details.

=COLOUR (read the colour assignment)

c=C0L0UR(index)

The COLOUR function takes an index number from 0 to 31s and returns the coour va1ue whi ch has been prev i ous1y ass igned to i t«

"Index" is simply the colour number whose shade you wish to determine. You can use this function to produce a list, of the current colour settings of your Amiga like so:

```
        For C=0 To 15
          Print Hex$(Colour(C),,3)
        Next C
```

PALETTE (set the current screen colours) 63

PALETTE list of colours

The PALETTE instruction is really just a rather more powerful version of COLOUR, Instead of loading the colour values one at a time, the PALETTE command allows you to install a whole new palette of colours in a single statement.

However you don't have to set all the colours in the palette at once. Any combination of colours can be loaded individually ;

```
        PALETTE *166,*5OO,t36d r. F5om S«,t« j...s-t Ih,<,» <<,1o...'«
```

You can also change selected colours in the middle of your list 5

```
        PALETTE $200,,$400 s Rem Change colours 0 and 2
```

It's important to realise that only the colours in the palette which are specifically set by this command will actually be changed. All other colours will retain their original values. Here *Ara* some examples:

```
        Palette 0,*FOOs*OFO
        Palette 0,$770
        Palette 0,,f>66
        Palette 0?*l?*2,$3?*a?*5,$6.!*?J1*8,,$9,,$A,1$B,,$CJ$D,,$E!1$F
```

At the start of your program the colour palette is automatically loaded using a list of default color values. These settings can be adjusted using a simple option from the AMOS configuration program.

This command can also he used to set the colours used by the Half-Bright and HAM modes. These extend the existing colour palette to generate dozends of extra colours on the screen™ See chapter 10...

!.ine drawing commands

GR LOCATE (position graphics cursor)

```
GR Locate x,,y
```

This sets the position of the graphics cursor to screen coordinates x,y. The graphics cursor is used as the default starting point for most drawing operations. So if you omit the coordinates from commands such as PLOT or CIRCLE, the objects will be drawn at the current cursor position. For example:;

```
        Gr Locate 10.,10 :: Plot ,
        Gr Locate 100,100 s Circle ,,100
```

=XGR (return x coordinate of gfx cursor) 64
=YGR (return y coordinate of gfx cursor)

```
x=XGR
y=YGR
```

These functions return the present coordinates of the graphics cursor;

```
        Circle 10,100,,.1.00
        Print Xgr.,Yqr
```

PLOT (plot a single point)

```
PLOT x?y [,cl
```

The PLOT command is the simplest drawing function provided by AMOS [illegible] co1oiir wi11 now be used in a11 subsequent drawi('sg operations..

If the co1our "<:" is omi11ed from this instruc11on, the point wi11 be plotted in the current colour. For example:

```
        Curs Off: Flash Off : Randomize Timer
        Do
          Plot Rnd(319),Rnd(i99),Rrid(i5)
        Loop
```

It's also possible to omit the X or Y coordinates from this instruction. The poin t wi11 be p1o11ed at the gf x cursor position .

```
        Plot 100,100,4
        plot ,,i5o
        Cls s Plot „
```

POINT (get the colour of a point)

```
c=P0INT(x,y)
```

POINT returns the colour index of a point at coordinates x?y ;

```
        Plot 100 J.100
        Print "The colour at 100,100 is ";Point(100,100)
```

DRAW (draw a line)6

DRAW is another very Basic instruction. Its action to draw a simple straight line on the Amiga's screen.

```
        DRAW xl,yl TO x2,y2
```

Draws a line between the coordinates x.1.,yl and *x2fy2*

```
        DRAW TO x3,,y3
```

Draw a line from the current gfx crsr position to x3?y3,. Examples :

```
        Colour 4,$707sink 4 ...
        Draw 0,50 To 200,50
        Draw To 100,5.00
        Draw To 0,50
```

BOX (draw a hollow retangle)

```
BOX xl,yl TO x2?y2
```

The BOX coramand draws a hollow retangular box on the screen., x1,y3 .^e the coordinates of the top left corner of the box, and x2,y2 are'the coordinates of the point diagonally opposite.

POLYLINE (multiple line drawing)

POLYLINE is *very* similar to DRAW except that it draws several lines at
[illegible] hollow [illegible]
single statement.

```
        POLYLINE x1,y1 TO x2,y2 TO x3,y3
```

CIRCLE (draw a hollow circle)

CIRCLE *x,y,r*

The Circle command draws a hollow circle with radius r and centre x,y.

As normal, if the coordinates are omitted from this command, the circle will be drawn from the current cursor position.

```
        Plot 100,100 : Circle ,,50
```

ELLIPSE (draw a hollow ellipse)

ELLIPSE *x,y,r1,r2*

The ELLIPSE instruction draws a hollow ellipse at coordinates *x,y.* The horizontal radius is r1. It corresponds to exactly half the width of the ellipse. *r2.* is the vertical radius and is used to set the height of the ellipse. The total height of the ellipse is r1*2

Line types

AMOS Basic allows you to draw your lines using a vast range of possible line styles.

SET LINE (set the line styles)

SET LINE mask

The SET LINE command sets the style of all lines which *are* subsequently drawn using the DRAW, BOX and POLYLINE commands.

"Mask" is a 16-bit binary number which describes the precise appearance of the line. Any points in the line which *are* to be displayed in the current ink colour are represented by a one, and any points which *are* to be set to the background colour are indicated by a zero. So a normal line is denoted by the binary number %1111111111111111. and will be displayed as______________. Similarly, a dotted line like............_ will be produced by a mask of *%1*111000011110000.

By setting the line mask to values between 0 and 65535, it is possible to generate a great variety of different line types ;

```
        Set Line $F0F0
        Box 50,100 To 150,150
```

This line style as no effect on shapes drawn with CIRCLE or ELLIPSE,

Filled shapes

PAINT (contour fill)

* PAINT jc,y.,mode

The PAINT command allows you to fill any region on the screen with a solid block of colour™ Additionally you can select a fill pattern for your shapes using the SET PATTERN command,.

x,y *Are* the coordinates of a point in*side the area to be filled. "Mode" can be set to either 0 or 1. A value of 0 terminates the filling operation at the first pixel found with the current border colour. A mode of 1 halts the filling operation at any colour which is different fromtheexistinginkco1our.

See EXAMPLE 6.1 in the MANUAL folder for a demonstration,.

BAR (draw a filled rectangle)

BAR xl,yl TO x2.,y2

Draws a filled bar from xl,yi -the coordinates of the top left corner of the bar- to x2,y2 -the opposite corner coordinates.

POLYGON (draw a filled polygon)

POLYGON xl, TO x2.,y2 TO x3,y3 ...
POLYGON TO xl,,yl TO x2,y2 ...

POLYGON generates a filled polygon in the current ink colour It's basically just a solid version of the standard POLYLINE command. There's no real limit to the number of coordnate pairs you may use,, other than the maximum line length permitted by AI1QS Basic: (255 chars).

Fill types

In AMOS Basic: you ' re not j ust restr i cted to f i 11 ing your shapes wi th a solid block of colour,, There *Are* dozens of fill patterns to choose froffl, and you can even load your own patterns directly from the sprite bank.

SET PATTERN (select fill pattern)

SET PATTERN pattern

This command allows you to select a fill pattern for use by your drawing operations. There *are* three possibi.1.ities

Pattem-=Q

This is the default, and fills your shapes with a solid block of the current INK colour.

If the pattern number is >0, AMOS Basic selects on of 34 built-in fill styles. These are found in the MOUSE.AM file? on your start-up disc:,, and can be edited using the AMOS Basic sprite definer,, Note that the first three images in this files are required by the mouse cursor (see CHANGE MOUSE). The fill patterns &re stored in the images from four onwards.

    Pattern<0

This is the most powerful option of all. "Pattern" now refers to a sprite image in bank one,. The image is number calculated using the formula: SPRITE IMAGE = PATTERN * (-1)

The selected image will be automatically truncated before use,, according to the following rules

% The width of the image will be clipped to sixteen pixels
* The height will be rounded to the nearest power of two, ie 1,,2..,64

Depending on the type of your image., the pattern will be drawn in one of two separate ways. Two-colour images are drawn in "monochrome". The actual colours in your image are completely discarded, and the pattern is drawn using the current ink and paper colours,,

It's also possible to produce multi-coloured fill patterns. In this case the foreground colours of your image and merged with the current ink colour using a logical AND,, Similarly the paper colours of your pattern is OR'ed with the sprite background (colour zero). If you wish to use your original sprite colours, you'll need to set the ink and background colours like son

```
    Ink 31,0
```

Don't forget to load your sprite palette from the sprite bank with GET SPRITE PALETTE before using these instructions,, otherwise the display is likely to look rather messy. Examples of this instruction can be found in EXAMPLE 6.2 in the MANUAL folder.

SET PAINT (set / reset outline mode) 70

```
SET PAINT n
```

Toggles the outline drawn by the POLYGON or BAR instructions. As a default this mode is set to OFF.

If n=l then outline mode will be activated.

Writing styes

GR WRITING (ghange writing mode)

GR WRITING bitpattern '

Whenever you draw some graphics on the screen,, you naturally assume that anything underneath it will be overwriteen. The GR WRITING command allows you to choose from a [illegible]
These can used to generate dozens of intriguing effects.

"Bitpattern" holds a sequence of binary bits which specify which graphics mods you wish to use,, Here's a list of the various possibilities along with a brief explanation of their effects;:

JAM! mode (Bit 0=0)

JAM1 only draws the parts of your graphics which are set to the current INK colour. Any sections drawn in the paper colour are totally omitted- This is particularly useful with with the TEXT command as it allows you to merge your text directly over an existing screen background., For examples

```
        Ink 2,5sText 140,80,"Normal Text":iGr Writing 0:
                                    Text 140,71,"JAM1"
```

JAM2 mode (Bit 0=1)

Thisisthedefau11condition,,Anyexisting graphics on the screen will he completely replaced by your new image,.

XOR mode (Bit 1=1)

XOR combines your new graphics with those already on the screen using a logical operation known as exclusive OR. The net result is to change the colour of the areas of a drawing which overlap ars existing picture.,

One interesting side effect of XQR mode is that you can erase any object from the screen by simply setting XOR mode and drawing your object aqain at exactly the same position. EXAMPLE 6.3 contains a simple demonstration of this technique and produces a neat rubber and banding effect.

INVERSEVID (Bit 2=1)

This reverses the image before it is drawn. So any sections of your image drawn in the ink colour will be replaced by the current paper colour and vice-versa,, INVERSEVID mode is often used to produce inverted text.

Since these modes are set using a bitpattern, it's possible to combine several mode together.

```
        Gr Writing 4 + 1 x Rem set JAI12 and INVERSEVID
        Or Writing 7   s Rem chooses JAM2,INVERSEVID and XOR
        Ink 2,5 ". Text 140,80,"Accession & Image rulez i"
```

NOTE: This command only affects drawing operations such as CIRCLE, BOX and graphical text (TEXT)., The drawing mode used by normal text commands like PRINT and CENTRE is set using a separate WRITING command,, See also AUTOBACK.

CLIP (restrict all gfx to a section of the screen)

CLIP Cxi!1yl TO x2,y2]

The CLIP instruction limits all drawing operations to a rectangular region of the screen specified by the coordinates xi,yl to x2,,y2,

xl,yl represent the coordinates of the top left hand corner of the rectangle, and x2,y2 hold the coordinates of the bottom right corner.

Note that it's perfectly acceptable to use coordinates outside the normal screen boundaries,, All the clipping operations will work as expected, even if only a section of the clipping rectangle is actually visible.

As you can see, only the parts of the circle which lie within the clipping rectangle have been drawn on the screen. The clipping *zone* can be restored to the normal screen area, by omitting all the coordinates fromthisinstruction.

See EXAMPLE 6.4 in the MANUAL folder.

Advanced techniques

SET TEfSPRAS (set temporary raster)

SET TEMPRAS [address,size]

This instruction allows experienced Amiga programmers to fine tune the amount of memory used by various graphics operations. WARNING; improper use of this instruction can crash your Amiga copletely!

Whenever an AMOS program performs a fill command,, a special memory area is reserved to hold the fill pattern,, This memory is automatically returned to the system after the instruction has been terminated. The size of the memory buffer is equivalent to a single bit plane in the current screen mode,, So the default screen takes up to a total of 8k.

The size and location of the graphics buffer can be changed at any time using the SET TEMPRAS instruction,

"Size" is the number of bytes you wish to reserve for your buffer-area. It ranges between 256 and 65536,,

The amount of memory required for a particular object can be calculated in the following way:

-• Enclose the object to be drawn with a rectangular box
-- The area required will given bys Size-=Width/3 * Height,,

If you are intending to use the PAINT command,, you should take care to ensure that your figure is tdosed*,, otherwise more memory will he neede and the system may crash.

"Buffer" can be either an addess or a memory bank,, The memory you reserve for this buffer should always be CHIP ram,, Since the buffer-area is now allocated once and for all at the start of your program,, there's no need to continually reserve and restore the memory buffer. This can speed up the execution of your programs by up to 5 %,,

You. can restore the buffer *area,* to its original value by calling the SET TEHPRAS command with no parameters.

See the EXAMPLE 6,5 in the MANUAL folder,, tot .

GOTO (jump to a new line number)

The action of GOTO is to transfer the control of the program one place to another. There *ar&* three forms of the GOTO command allowes in AMOS;

GOTO label

"label" is an optional place marker at the side of a line. Label names are defined using the ";" colon character like sos

```
label;
```

The label name can consist of any string of alphanumeric characters you like, including "-". It's constructed using the same rules which apply for variables and procedure names,

GOTO line number

Any AMOS Basic line cars be optionally preceded with a number,. These line numbers *&re* included solely for compatibility purposes with other versions of Basic (such as Si TOES for the Atari ST). It's better to rely on labels instead, as these are much easier to read and remember.

GOTO variable

Variable can be any allowable AMOS Basic, expression, This expression may be either a normal ingerer or a string. Integers run a line number for your GOTO, whereas strings hold the name of a label.

Technically speaking,, this construction is known as a computed goto. It's generally growned upon by serious programmers, but it can be incredibly useful at times. Examples:

```
R00M=3
BEGIN:
Goto "ROOM"-J-StHKROOM)-" ""
E n d
R00M3;
Print "Room three!"
Goto BEGIN
```

GOSUB (jump to a subroutine) " 7 4

GOSUB is another outmoded instruction, and provides you with the useful ability to split a program into smaller, more manageable chunks, known as subroutines. Nowadays, GOSUB has been almost entirely supplanted by AMOS Basic's procedure system. However, GOSUB does form a useful half-way house when you're converting programs from another version of Basic: such as STOS,,

As with GOTO,, there *&re* three different forms of the GOSUB instruction. .

QO8US n	Jump to the -,r,ubr<:,«tin«» «t line r,
GOSUB name	Jump to an AMOS label

GOSUB exp   Jump to a label or line which results from the
               expression in "exp"..

Example;

```
For I=1 To 10
  Gosub TEST
Next I
Direct
TEST;
Print "This is an example of GOSUB";Print "I equals ";I
Returns Rem Exit front subroutine TEST and return to main prg.
```

It's good practice to always plave your subroutines at the end of your main program as this makes them easier to pick out from your program listings. You should also add a statement like EDIT or DIRECT to end of your main program, as otherwise AIIOS may attempt to execute your GOSUBs after the program has finished, generating an error message-

RETURN (return from a subroutine)

RETURN

RETURN exits from a subroutine which was previously entered using GOSUB. It immediately jumps back to the next Basic instruction after the original GOSUB.

Note that a single GOSUB statement can contain several RETURN commands., So you can exit from any number of different points in your routine depending on the situation.

POP (remove the RETURN info after a GOSUB)

POP

Normally it's illegal to exit from a GOSUB statement using a standard GOTO.. This can occassionally be inconvenient., especially if an error occurs, which makes in unacceptable to return to your program from the precice point you left it.

The POP instruction removes the return address generated by your GOSUB, and allows you to leave the subroutine in any way you like, without first having to execute the final RETURN statement. Example:

```
Do
  Gosub TEST
Loop
BYE:
Print "Popped Out"
Directs
TESTs
Print "Hi there!"
If Mouse Key Then Pop :: Goto BYE
Return
```

The IF...THEN instruction allows you to make simple decisions (within a Basic program. The format is:

```
IF conditions THEN statements 1 [ELSE statements 2]
```

"conditions" can be any list of tests including AND and OR,, Statements 1 and Statements 2 must be a list of AMOS Basic instructions,, If you want to jump to a line number or a label,, you'll have to include a separate GOTO command like sos

```
        If test Then Goto Label s This is fine.
```

If you forget about this, and leave the "Goto", you'll get an error message "procedure not defined".

```
        If test Then Label s Rem THIS CALLS A ^PROCEDURE*
```

The scope of this IF...THEN statement is limited to just a single line of your Basic program. It has now been superceded by the much more powerful IF...ELSE...ENDIF command.

IF...[ELSE]...ENDIF (structured test)

Although the original form of IF...THEN is undoubtedly useful, it's rather old fashioned when compared with the facilities found in a really modern version of Basic such as AMOS. This allows you to execute whole lists of instructions depending on the outcome of a. single test.

```
        IF tests=TRUE
          <List of statements 1>

        ELSE
          <List of statements 2>

        ENDIF
```

Note; it's illegal to use a normal IF...THEN inside a structured test! These should be replaced by their equivalent IF...ENDIF instruction ;

```
        If test Then Goto Label Else Label2
```

This now becomes:

```
        If test :: Goto Label :i Else goto Label2 s Endif
```

or ;

```
        If test
          Goto Label
        Endif
```

Here is an example of the IF.,,ENDIF statement in actions

```
        Input "Enter values for a,b and c";A,B,,C
        If A=B
          Print "Equal"
        ELSE
          Print "Different";
          If AOB and AOC
```

```
            Print % and C is not the same too!"
          Endif
        End if
```

Each IF statement in your program MUST be paired with a single ENDIF' command as this informs AMOS Basic precisely which group of instructions *are* to be executed inside your test,.

Nate that "THEM" is not used by this form of the instruction at all. This may take a little getting used to it you are already experienced with one of the other versions of Basic for the Commodore Amiga.

FOR...NEXT (repeat a section of a code a specific number of times) 77

```
FOR index=first TO last [STEP inc]
•Gist of instructions!?-

NEXT [index]
```

"Index" holds a counter which will be incremented after each and every loop. At the start of the loop,, this counter will be loaded with the result of the expression "first".. The instructions between FOR and the NEXT *Are* now performed until the NEXT is reached,

"inc" is a value which will be added to the counter after each loop by the NEXT instruction. If this is omitted,, the increment will be automatically set to 1.

Note that if "inc" is negative., the loop will be halted when the counter is less than the value in "first". So the entire loop will be performed in reverse.

Once inside loop, "index" can be read from your program just like a normal variable. But you *are* *M0T* allowed to change its value in any way as this will generate an error message.

EachF0Rstate*ms*ntin*ya*urprogramMUSTbematchedbyasing1eNEXT instruction. You can't use the shorthand forms found in other Basics like NEXT R1,R1. Here *ures* a couple of examples of these loops:

```
        For 1=32 to 255 s Print Chr*(I);sNext I

        For Rl=20 to 100 Step 20
          For R2=20 to 100 Step 20
            For A=0 To 3
              Ink A
              Ellipse i60.i00,,Rl.,R2
            Next A
          Next R2
        Next Rl
```

WHILE...WEND (repeat a section of code while a condition is true) 78

```
WHILE condition

list of statements

WEND
```

"condition" can be any set of tests you like and can include the constructions AND, OR and MOT., A check is made on each turn of the loop. If the test returns a value of ~1 (true),, then the statements between the WHILE and WEND will be executed, otherwise the loop will be aborted and Basic will proceed to the next insturction. Type the following example:

```
Input "Type in a number"5X
Print "Counting to 1.1"
While X<11
  Inc: X
  Print X
Wend
Print "Loop terminated"
```

The number of times WHILE loop in this program will executed depends on the value you input to the routine, If you enter a number larger than 10, the loop will never be executed at all. WHILE will therefore only execute the statements if the condition is TRUE at the start of your program.

REPEAT...UNTIL (repeat until a condition is satisfied)

```
REPEAT
 :  :
list of statements
 :  :
UNTIL condition
```

REPEAT...UNTIL is similar to WHILE...WEND except that the test completion is made at the end of the loop rather than the beginning. The loop will be repeated continually until the specified condition is FALSE. So it will always be performed at. least once in your program. Example:

```
Repeat
  Print "AMOS Basic"
Until Mouse KeyOO
```

DO,,..LOOP (loop forever)

```
» o
:  :
list of statements
:  :
LOOP
```

The DO...LOOP commands take a list of Basic: statements and repeat them continually. In order to exit from this loop, you'll need to use a special EXIT or EXIT IF instruction.

The advantage of this system is that it's a structure alternative to the GOTO loops that tend to crop up in earlier versions of Basic. Take the following example:

```
TEST:
Input "Another game (Y/N)"$AW*
If Upper*(AN$)-"N" Then Goto BYE
```

```
        GAME : Rem call play game procedure
        Goto TEST
        BYE:
        End
```

Now a second version using DO.., ..LOOP

```
             INput "Another game (Y/N)";AN*
             Exit If Upper*(AN*)="N"
             GAME : Rem call play game procedure
           Loop
           End
```

EXIT (Exit from a DO......LOOP)

EXIT [n]

The EXIT command exits immediately from one ore (nore program loops created with the FOR...NEXT, REPEAT .... UNTIL..., WHILE, ... WEND,, or DO,,.,.LOOP statements. Your AMOS program will now jump directly to the next instruction after the current loop.

"n" is the numver of loops you wish to leave. If it's omitted, then only the innermost loop will be terminated.

EXIT IF (Exit from a loop depending on a test) 80

EXIT IF expression!! ,,n]

"expression" consistes of a series of tests in the standard AMOS format. The EXIT will only be performed if the result evaluates to -1.

The "n" parameter works the same way as using EXIT command.

EDIT (stop running the prog and return to Editor)

EDIT

The EDIT directive stops the current program and returns to the AMOS Basic: editor., This can be very useful when you *are* debugging one of your progs.

DIRECT (exit to direct mode)

DIRECT

Terminates your program and jumps to the direct mode immediately. You can now examine the contents of your variables or list your programs out to the printer.

END (Exit from the program) 81

END

This instruction exits from a program.. You'll now be given the option to return to either the editor or to direct mode.

ON...PROC (jump to one of several procedures depending on a variable)

ON v PROC proci., proc:2,, proc3, . ..procN

Jumps to a named procedure depending on the contents of variable v. Note that any procedures you use in this command CANNOT include parameters. If you need to transfer information to this procedure, you should place them in *global* variables instead,, See PROCEDURES for'a full explanation of this technique,,

The ON,, ..PROC command is effectively equivalent to the following!

```
If v=J. Then Proci
If v:=2 Then Proc2
  :     :
If v=n Then ProcN
```

ON ...GOTO (jit dip to one of a list of lines depending on a variable)

ON v GOTO linel, I:i.ne2., I:i.ne3, n..l:i.neN

The OK! GOTO instruction lets your program jump to one of a number of lines depending on the result of an expression in v. It's equivalent to the following lines:

```
If v=l Then Goto Linel
If v=2 Then Goto L.ine2
  :     :
If v=n Then Goto LineN
```

ON...GOSUB (GOSUB one of a list of routines dependig on var)

ON *v&r* GOSUB linel, Iine2, Hne3, „. „

This is identical to ON.,,, .GOTO except it uses a gosub rather than a goto to jump the line,,

LVfcRY n GOSUB (call a subroutine at regural intervals)

EVERY n GOSUB label

The ON EVERY statement calls the subroutine at label at regural intervals, without interfering with your main program.

"n" is the length of your interval in 50ths of a second. The time taken for your subroutine to complete must always be less than thi-> period, or you'll get an error.

Error handling

ON ERROR GOTO (trap an error within a Basic prog)

ON ERROR GOTO label

This command allows you to detect and correct the errors inside an AMOS Basic program, without having to return to the editor window., Sometimesj errors can arise in a program which &re impossible to predict in advance. Take, for instance, the following routines

```
        Input "Enter two numbers"?A,B
        Print A;" divided by "nBs" is ";A/B
      Loop
```

This program vjorks fine until you try to enter a zero for B. You can avoid the "division by zero error" by trapping the error with an ON ERROR GOTO instruction like so;

```
      ON ERROR GOTO label
```

Whenever an error occurs in your Basic program, AMOS will now jump straight to "label". This will be? the starting point of your own error correction routine which can fix the error and safely return to your main program.

Note that error handler MUST exit using a special RESUME instruction. You are not allowed to jump back to your program with a normal GOTO statement.

```
      On Error Goto HELP
      Bo
        Input "Enter two numbers" i;A,B
        Print A;" divided by "5B5" is ";A/B
      Loop
      HELPs
      Print s Print s Bell
      Print "I'm afraid you've attempted to divide with zero!"
      Resume Next; Rem Return back to the next instruction.
```

In order for this system to work, it's essential that an error does not arise inside your error correction routine, otherwise AMOS will halt your program ignominiously.

The action of ON ERROR GOTO can be disabled by calling ON ERROR with no parameters.

```
      On Error !i Rem Kill error traps
```

OKI ERROR PROC (Trap an {error using a procedure)

ON ERROR PROC name

Selects a procedure which will be called automatically if there's an error in the main program,, It's really just a structured version of the oKI in: p ft o a a a r o -_1 ^ -t e « r, -t..

Although your procedure must be terminated by and END PROC in the normal way,, you'll need to return to your main program with an additional call to RESUME!. This can be placed just before the final END PROC statement.

RESUME (resume execution of the program after an error)

There *Are* five possible formats of this instructions

```
RESUME
```

Jumps back to the statement which caused the error and tries again»

```
RESUME NEXT
```

Returns to the instruction just after the one which caused the error,

```
RESUME LINE . . . . . . .
```

Jumps to as specific line point in your main program, "line" can refer to either a label or a normal line number. This may *M0T* be used to re-enter a procedure!

Procedures are? treated slightly differently. If you want to jump to a particular label,, you have to place a special marker somewhere in the procedure you are checking for errors. This may be accomplished using the RESUME LABEL command. There *&rs* two separate versions.

```
RESUME LABEL label
```

Defines the label which is to be returned after an error. This must be called outside your error handler just after the original ON ERROR PROC or ON ERROR GOTO statement.

```
RESUME LABEL
```

Used inside your error handler to jump straight back to the label you've set up with the previous command. Examples

```
On Error Proc HELP
Resume Label AFTER
Error 12
Print "Never Printed"
AFTER s Print "I've returned here"
End
Procedure HELP
  Print "Oh Dear, I think there's an error!"
  Resume Label
Endproc
```

=ERRhl (return the number of last error)

e=ERRN

If you're creating your own error handling routines using the ON ERROR command, you'll need to be able to check precisely which error has occurred in the main program.

When an error occurs,, ERRN is automatically loaded with its identification number., See the Apeendix at the end of this manual for a

*~ full list of the possible errors.

```
        Print ERRKi
```

ERR0R (generate an error and return to the Edi tor)

ERROR n

The action of the ERROR command is to actually generate an error,, Supposing you have created a nice little error handling routine which is able to cope with all possible disc errors. ERROR provides you with a simple way of simulating all the various problems, without the inconvenience of the actual error. Examples

```
        Error 40
```

Quits the program and prints out a "Label not defined" error.

```
        Error Errn
```

This uses the ERRN function to print the current error condition after a problem in your program.

1 Text Attributes

PEN (set colour of text)

PEN index

The PEN instruction sets the colour of all the text which will be displayed in the current window. This colour can be chosen from one up to 64 different possibilities depending on the gfx mode you're using. Examples

```
          PEN 6 .
```

=-PEN$(n) (change the pen colour using ctrl chrars)

a$=-PEN$(n)

PEN$ returns a special control sequence which changes the pen colour insidea.string.Thenewpencolourwi11beautomatica11yassigned whenever this string is subsequently printed on the screen,, Examples

```
        C*=Pen*(2)+"White "+P»n*(6)+"Blue"
        Print C$
```

The string returned by PEN$ is in the format.:

```
 Chr*(27)+"P"+Chr*(48+n)
```

PAPER (set colour of the text background)

PAPER index

"index" can be a number between 0-63.

=PAPER$(n) (return a control sequence to set the paper colour) 88

x$=PAPER$(index)

PAPER* returns a character string which automatically changes the background colour when it's printed on the screen. For example:

```
        Pen 1: C*=-Paper*(2)+"White "+Paper$(6)+"Blue"
        Print C$
```

INVERSE ON/OFF' (enter inverse mode)

INVERSE ON/OFF

The INVERSE command swaps the text and *ho b«k,round col<ours.

SHADE ON/OFF (shade all subsequent text)

SHADE ON/OFF

SHADE ON highlights your text by reducing the brightness of the characters with a mask pattern,, The shade of your text can be returned to normal using SHADE OFF

UNDER ON/OFF (set underline mode)

UNDER ON/OFF

UNDER ON underlines your text when it's printed on the screen, UNDER OFF turns off the mode,

WRITING (change text writing mode)

WRITING w1 [,w2]

The WRITING command allows you to change the writing mode used for all subsequent text operations. This determines precisely how your new text will be combined with the existing screen data.

w1=0	REPLACE (Default)	Your new text will obliterate anything underneath it,
w1=1	OR	Merges the characters onto the screen with a logical OR.
w1=2	XOR	Chars are combined now with XOR.
w1=3	IGNORE	Printing operations are ignored!

The secont number chooses which parts of the text will be printed on the screen. This option can be omitted if required.

w2=0	Normal	The text is output to the screen along with the background.
w2=1	Paper	Only the background of the text is drawn on the screen,
w2=2	Pen	Ignores the paper colour and writes the text on a background of colour zero.

Do *N0T* confuse this with GR WRITING!

Cursor functions

AMOS includes a range of facilities which let you move cursor to any part on the screen.

LOCATE (position the cursor)

LOCATE x,y
LOCATE x,
LOCATE ,y

Locate moves the text cursor to the coordinates x,y. This sets the starting point for all future printing operations. All screen positions are specified using a special set of text coordinates,, These &"^ meadured in units of a single character relative to the top left corner of the text window, For instance the coordinates 15,10 refer to a point 10 chars down and 15 to the right,,

If you attempt to print something outside window limits an error will be generated.

Note that the current screen is always treated as window 0. So you don't have to actually open a window before using one of these functions,,

CI10VE (relative cursor movement)

CMOVE w,h

Moves the cursor a fixed distance away from its present position. If your cursor was at 10,10,, then typings

```
        CMOVE 5,-5
```

would move the cursor to 15,5. Like LOCATE you can omit either one of the coordinates as required.

=AT (return a sequence of ctrl chars to position the cursor) 91

x*=AT(x,y)

The AT function allows you to change the position of text directly from inside a character string,, It works by returning a string in the format;

```
          Chrf(27)+"X"+Chr*(27)+"Y"+Chr$(48+Y)
```

Whenever this string is printed, the text cursor will he moved to the coordinates x,y» For example:

```
      A*="This"+At(10,i0)+"Is"+At(l,2)+"The Power Of "+At(20,20)+"AilOS i"
      Print A$
```

These AT commands &re perfect for hi-score tables as they allow you to position our text once and for all during your prog rains initialisation phase. You can now update the score at the correct, point on the screen using a single print statement... Here's an examples

```
        HI_SC0RE*=At(20,10)+"Hi Score "
        SC0RE-10OO0
        Print HI_SC0RE$;SCORE
```

Conversion functions

AMOS Basic provides you with four useful functions which readliy enable you to convert between text and graphics coordinates..

=XTEXT(convert an x coordinate gfx->text format) 92

=YTEXT (convert an y coordinate gfx-->text format)

t^XTEXTCx)
t=YTEXT(y)

These functions take normal x/y coordinates and convert them to text coordinates relative to the current window. If the screen coordinate lies outside this window then a negative value will be returned. See EXAMPLE 8.1.

=XGRAPHIC (convert an x coordinate text-->gfx format)
=YGRAPHIC (convert any coordinate text->gfx format)

g=XGRAPHIC(x)
g=XGRAPHIC(y)

These functions *are* effectively the inverse of XTEXT and YTEXT in that they take a text X (or) Y coordinate ranging from 0 to the width/height of the current window and convert them to absolute screen coordinates. See EXAMPLE 8.2

Cursor coinmands
================

The text cursor serves as a visible starting point of all future text operations„ 11's usua11y displayed as a f1ashing horizontal bar, although this may be changed using the SET CURS and CURS OFF commands,

By moving the cursor on the screen, you can position your text practically anywhere you like. Remember,, all coordinate measurements are taken using TEXT coordinates relative to the current window.

HOME (cursor home)

HOME

Moves the text cursor to the top left hand corner of the current window (coordinates 0,0)

CDOWN (cursor down)

CDQWN

Pushes the text cursor down by a single line.

=CDOWN* (return a Chr$(31) character)

x*=CD0WN*

CDOWN* is a function which returns a special control character which automatically moves the cursor when it is printed. So Print CDOWN* is identical to CDOWN. CDOWNf allows you to combine several cursor movements in a single string. For examples

```
c * - - "viucdo™,,*
For A=0 to 20
  Print C*5
```

Next A

CUP (cursor up)

CUP

Moves the text cursor up a line in the same way that CDOWK! moves down.

-•CUP* (return a Chr*(30) character) _.

x*=CUP*

CUP* returns a control string .which moves the cursor up by a single character.

CLEFT (cursor left) . 94

CLEFT

Displaces the text cursor one character to the left,

=CLEFT* (Control string for CLEFT Chr*(2?))

x*»CLEFT*

Moves the text cursor one character left. Works like =-CUP*.

CRIGHT (cursor right)

CRIGHT

Moves cursor one place to the right.

=CRIGHT$ (Generate a Chr*(28) control string for CRIGHT)

x*=CRIGHT*

Is the opposite of CLEFT*.

XCURS (return the X coordinate of the text cursor)
YCURS (return the Y coordinate of the text cursor) 95

x=XCURS
y=YCURS

XCURS is a variable containing the current X coordinate of the text cursos (in text format)™ YCURS holds the Y coordinate of the cursor.

SET CURS (set text cursor shape)

SET CURS Ll,L2,L3,L4,L5,L6,i..7,L8

This instructoin allows you to change the shape of the cursor to anything you like. The shape is specified by a list of bit-patterns held in the parameters L1-L8, Each parameter determines the appearance of the horizontal line of the cursor,, luunbered from top to bottom.

Every bit represnts a single point in the current cursor line. If it's set to 1 then the point will be drawn using colour number 3 - otherwise it will be displayed in the current PAPER colours,, Example:

```
        Ll=Sill11111
        L2=211111110
        L3=X1ill1100
        L4=:illlllOOO
        L5=X11110000
        L6=y.l1100000
        L7=£11000000
        L8=?U0000000
        Set Curs LI ,1.2,13,L4,L5,L6,L?,LS
```

CURS ON/OFF (enable/disable text cursor)

CURS OK!	makes text cursor visible
CURS OFF	hides the cursor in current window

MEMORIZE X/Y (save the X or Y coordinates of the text cursor)

MEMORIZE X
MEMORIZE Y

The Memorize commands store the current cursor position,.

REMEMBER X/Y (restore the X or Y coordinate of the text cursor)

REMEMBER X
REMEMBER Y

REMEMBER positions the cursor at the coordinates saved by a previous call to MEMORIZE„ If MEMORIZE has not been used then the coordinates will be set to zero. See EXAMPLE 8.3

CLIME (clear part or all of the current cursor line)

CLIME En]

Clears the line on which the cursor is positioned. If n is present then "n" characters &r& cleared starting at the current cursor position,.

CURS PEM (choose a new colour for the text cursor)

CURS PEW n

Changes the colour of the text cursor to index number n.

Text Input/Output
=================

CENTRE (print a line of text centred on the screen)

CENTRE a*

Takes a string of characters in a* and prints it in the centre of the screen. This text is always output on the current cursor line.,

```
        Locate 0,1
        Centre "This is a centered TITLE"
```

=TAB* (print tabulation)

x$=-TA6*

TAB* returns a control character known as a TAB (Ascii 9), When this character is printed the text cursor will be immediately moved several places to the right- The size of this movement can be set using the SET TAB kommand. As a default., the tab spacing is set to four (4),,

SET TAB (change the tabulation)

SET TAB n

This specifies the distance the text cursor will move when TAB character is printed.

REPEATS (repeat string)

x$=REPEAT*(a$,,n)

The REPEAT* function allows you to print out the same string of characters several times using a single PRINT statement,,

It works by adding a sequenve of control characters into variable X$. When this string is printed,, AIIOS simply repeats a* to the screen n times. Possible values for n range between 1 and 207. See EXAMPLE 8.4.

The format of the control string iss

```
        Chr$(27)+"RO"+A$+Chr$(27)+"R"+Chr$(48+n)
```

Advanced Text Commands

ZOhEt (set up a zone around a piece of text)

x$=ZONE$(a$,n)

The ZOK1E$ function surrounds a section of text with a screen zone. After you have defined one of these zones you can check for coillisions between the zone and the mouse using the ZONE function. This allows you to create powerful on-screen menus and dialogue boxes without having to resort to any complicated programming tricks.

a* is a string containing the text for one the "Buttons" in your dialogue box. This button will be activated automatically when you print x$ to the screen.

n specifies the number of screen zone to be defined. The **max.** number of these zones depends on the value you specified with RESERVE ZONE.

See the EXAMPLE 8,5 program in the MANUAL folder,, The format of the control string is:

```
Chr*(27)+"ZG"+A$+Chr*(27)+"R"+Chr*(48+n) ,
```

BORDER* (add a border to some text)

x$=BORDER*(a*,n)

This returns a string of control characters which instructs AMOS to draw a borcler aound the requi red tex t. 11' s common 1 y used in conjunction with the ZONE* command to produve the fancy buttons found in dialogue boxes and alert windows.

R is the border number ranging from 1 to 16 and a$ holds the text to be enclosed by the border. The text in a* will start at the current cursor position so don't be surprised when you get strange results printing at 0,0. To create a screen zone by a border try this:

```
Print Border$(Zone$(" CLICK HERE ",,1),,2)
```

This would enclose the text with zone number 1 and border 2» The control sequence isx

```
Chr$(27)+"EG"+A*+Chr*(27)+"R"+Chr$(48+n)
```

HSCROLL (horizontal text scrolling)

HSCROLL n

This scrolls all the text in the currently open window horizontally by a single character position, n can take the following values;

1 = Move current line to the left
2 = Scrolls entire screen to the left
3 = Move current line to the right
4 = Move screen to the right

VSCROLL (vertival scroll)

'-.'SCROLL n

Scrolls the text in the currently open window vertically.

1 = Any text at the cursor line and below is scrolled down
2 -- Text at cursor line or below is moved up
3 -- Only text from the top of the screen to the cursor line is scrolled up
°> -- Text from top of the screen to the current cursor position is s c rolled do wn

Blank lines are inserted

to pad out the gap left by the scrollingoperation.

Windows

The AMOS windowing commands allow you to restrict your text and graphics operations just a part of the current screen.

AMOS windows can be used with the zone commands to produce effective dialogue boxes such as file selectors and high score tables., A typical warning box, for instance, can be easily generated with just a couple lines of AMOS Basic.

WINDOPEN (create a window)

WINDOPEN n, x. y, w, h [,border [,set]]

The WINDOPEN instruction opens a window and displays it on the screen.. This window will now be used for all subsequent text operations,,

n is the number of the window to be defined,. AMOS allows you to create as many windows as you like,, limited only by the amount of available memory. As a default, window number zero is assigned to the current screen. So don't a11empt to re-open this window using WIND0PEN or change it with WIND SIZE or WIND MOVE.

x,y are the graphics coordinates of the top left hand corner of your new window., Since AMOS windows are drawn using the Amiga's blitter chip, the window area, must always lie on a 16-pixel boundary. In order to achieve this, the x coordinates are automatically rounded to the nearest multiple of 16. Additionally, if you've included a border for your window, the X coordinate will be incremented by a further eight. This will ensure that the working area of your window always starts at the correct screen boundary. There are no restrictions whatsoever on the *y* coordinates.

w,h specify the size in characters of the new window,, These dimensios must always be divisible by 2.

"border" selects a border style for your window,, There *AY'B* 16 possible styles, with values ranging between 1 and 16,,

Window borders can also include up to two optional title lines. One title is displayed along the top of the window and another may be added at the bottom.

AMOS windows may contain either text or graphics, just like the intuition system. Each window can be assigned it's own individual character set with the powerful WINDOW FONT command. There's also a powerful WIND SAVE instuction which saves the screen area inside your windows. Whenever you move one of these windows the contents underneath will be automatically redrawn. For example;

```
        For W=i To 3
          Windopen W,(W-l3*96,50,10,101
          Paper W+3 : Pen W+6 : CIw
          Print "Window ";W
        Next W
```

You can flick between these windows using the WINDOW command. Try typinq the following statements from the Direct modes

```
        Window 1 : Print "AMOS11
        Window 3 s Print "in action!"
        Window 2 : Print "Basic"
```

The active window can always be distinguished by a flashing cursor through this can be turned off using the CURS OFF- command if required,

WINDOW FONT (change window font)

WINDOW FONT n

Changes the font used by the current window to set n. n is the number of a graphics font which has been previously installed with the GET FONT command. This font Kmust* have dimensions of exactly 8x8, Proportional fonts are not allowed.

Since the window vborders make use of some of these characters, you fliayqet rather odd resu11s when yau're using standard WBench fonts.

WIND SAME (save the contents of the current window)

WIND SAVE

The WIND SAVE command allows you to move your windows anywhere on the screen without corrputing your existing display.

Once you've activated this feature,, any windows you subsequently open will automatically save the entire contents of the windows underneath„ This area will be redrawn whenever you close a window or move it to a new position™

It's important to note that this option saves the contents of the current window,, rather than the one you *are* defining with WIND OPEN-

At the start of your program the current window will be the default screen and wi11 take up a massive 32k of memory. If you wished to save the background underneath a dialogue box the most of this memory would be completely wasted.,

The solution is to create a dummy window of the required size, and to position it over the zone you wish to save. You can now execute a WIND SAVE command and continue with your program as normal,,

When you subsequently call up your dialogue box the *area* underneath will be saved as part of your dummy window. So it will be automatically restored after your box has been removed,,

BORDER (change the window border of the) current screen)

BORDER n,paper,pen

Tho BORDER command sets the border of the:- current window to styles? number n. This border is drawn using a group of characters installed in the default font,, It is therefore possible to create your own border styles using the font tie finer accessory.

The paper and pen options allow you to freely choose the colours of your border- Acceptable border numbers range from 1 to 16.

> Any of the parameters may be omitted from this instuction so **the** following commands are legals

```
BORDER 2,,
BORDER 2,,3
```

TITLE TOP (define the upper title for the current window)

TITLE TOP t$

This instruction sets the top line of the current window to the title string in t$. Only bordered windows may be titled in this way,

```
Windopen 5,1,1,20,10
Title Top "Window Number 5"
Wait Key
```

TITLE BOTTOM (define the lower title for thecv, rrentwindow)

TITLE BOTTOM b$

This command assigns the string b$ to the bottom title of the current window.

WINDOW (change current window) 102

WINDOW n

WINDOW activates the window n as the current window- If the automatic saving system has been initiated, this window be immediately redrawn along with any of its contents. See EXAMPLE 8.6 in the Manual folder.

=WIMDON (Return the value current window)

w=WINDOW

WINDON returns the identification number of the currently active window,,

WIND CLOSE (close the current window)

WIND CLOSE

[illegible] if you want the araathatwashiddenberedrawnby,.

WI.NDPI0VE(moveawindow)

WINDMOVE x,y •

Windmove moves the current window to graphics coordinates %,y. As with the original window definitions the x coordinate will be rounded to the nearest 16-pixel boundary,,,

WIND SIZE (change the size of the current window) 103

WIND SIZE sx,sy

This command changes the size of an AMOS window,, The new sizes,, sx and sy, are specified in units of a single character,, Sx must be divisible by two,, See EXAMPLE 3.7.

If you've previous!/ called the WIND SAVE command, the original contents of your window will be redrawn by this instruction. If the new window is smaller than the original one, any parts of the image which lie outside will be lost. Alternatively,, if you've expanded your window, the area around your saved region will be filled with the current paper colour. Also note that after a WIND SIZE command the text cursor is always reset to coordinates 0,0,,

CIW (clear the current window)

CLlti

Erases the contents of the current window and fills it with the current PAPER colour.

Slider bars 104'

AHOS incorporates three insturctions which allow you to display a standard slider bar on the screen,, These sliders cannot be manipulated directly with the mouse. In order to create a working slider bar, you'll need to write a small Basic routine to perform this operate in your main program. Due to the sheer power of the AMOS system, this is extreme1y easy to accofnp1ish, and the res u11s can be extrem e1y impressive,, as can be seen from EXAMPLE ELS.

HSLIDER (draw a horizontal slider)

HSLIDER xl,yl 0T x2,y2,total,pos,,size

Draws a horizontal slider bar from xl,yl to x2,y2. "total" is the number of individual units which the slider will be divided into. Each unit represents a single item in the object you are controlling with the slider. TSo in the editor window, "total" would be set to the number of lines in the current program. The size of each unit is calculated from the following formula;

(X2-X1)/Total

r

I '' ..DOC« iS thp position of the slider box from the start of the slider,
* measured in the units you specified using "total",, "size" is the lengtn
i - of the slider box in the previous units,, See tXftMHLb ti.y.

VSLIDER (draw a vertical slider)

```
VSLIDER xl,yl TO x2,y2,total,pas:,5ize
```

VSLIDER is almost identical to the previous HSLIDER insturction. It displays a simple slider from xl,,yl to x2,y2. See EXAMPLE 8.10.

SET SLIDER (sets the fill patterns used in a slider)

```
SET SLIDER bl,b2,b3,pb,sl>,s2,s3,ps
```

Although this command looks incredibly complicated, it's actually rather simple. SET SLIDER enters the colours and patterns to be used in the slider bars created with the H/VSLIDER commands.

"bl,b2,b3" set the ink,, paper and outline colours for the background of the box, "pb" chooses the fill pattern to be used for these regions. 105

"sl,s2,s3" input the colours of the slider box, and "sp" selects the pa11ernitistobefi11edwith.

"bp" and "sp" can be any fill patterns you wish,, As usual, negative value refer to a sprite image from the current sprite bank,, This allows you to create amazing colorful slider boxes.

Fonts

There *are* two different types of fonts available in AMOS - text fonts and graphic fonts. The text fonts *Are* those used by the PRINT and. WINDOW commands. Text fonts are known as character sets and each APiOS Basic window can have its own individual set. The graphic fonts *Are* much more flexible and offer a wider range of styles:

Graphic text

Your Amiga computer is capable of displaying an impressive variety of different text styles,, The original WorkBench disc was supplied with eight attractive fonts in a range of sizes, and many more of these fonts are freely available from the public domain- If you've upgraded to WorkBench 1.3, you'll also be able to design your own fonts usinq the FED program on the Extras disc.

Ail OS provides you with total support for these fonts. Text can be printed in any of the available typefaces at any point on the screen.

AMOS fonts can be used to add spice to even the most Basic games. These *Are* invaluable for producing the loading screens arid hi-score tables in your games,, So it's a good idea to make full use of them in your progs.

TEXT (print graphical text)

TEXT x,y,t*

TFXT prints a line of text in t* at graphical coordinates x,y. All coordinates are measured relative to the characters baseline. This can be determined using a special TEXT BASE function.

Normally the baseline is positioned at the bottom of the character, but some lowercase letters., such as "g", have a "tail" which extends slightly below this point.

As a default the type styles is set to eight-point Topaz,, This may be changed at any time using the SET FONT instruction. Try the following program and notice how text can be placed at any pixel position on the screen.

```
  Do
    Ink Rnd(15)+l,Rnd(15): Text Rnd (320H1 ,Rnd (198) + 1," AMOS Basic"
  Loop
```

Al so notice how the colour of your text is set with INK rather than the expected PEN and PAPER commands. This emphasizes the fact that the TEXT command is basically a graphical instruction. So the control sequences created by functions like CUF'$ will be printed on the screen instead of being correctly interpreted.

There is no automatic line feed when the text reaches the end of the current window. If you attempt to print something too large,, the text will be neatly clipped at the existing screen boundary. This can be seen by the example below:

```
        Print String*("A%100):Text 0,100,String*("A",100)
```

GET FONTS (create a list of all available fonts)

GET FONTS

The GET FONTS command creates an internal list of the all fonts avai 1 ab 1 e f ram the curren t star t-up d isc „ This 1 ist is essen tia 1 to the running of the SET FONT command, so you should always call GET FONTS at least once before attempting to change the present font setting. The contents of this list can be examined using the FONT* function.

WARNING! In order for GET FONTS to work,, your current AH0S work disc must always contain a copy of the standard LIBS folder along with its contents. It's important to remember this fact when you ars distributing run-only or compiled programs because unless your discs contain the required files, AMOS Basic will almost certainly crash!

GET DISC FONTS (create a list of the disc fonts)

GET DISC FONTS

This command is identical to the previous GET FONTS instruction except that it only searches for fonts on the disc. These fonts are contained in the FONTS [illegible] If you [illegible] your own fonts with AMOS basic, you'll need to copy these onto your normal start-up disc. See the manual supplied with your Amiga for'details of this procedure.

GET ROM FONTS (create a list of the rom fonts)

GET ROM FONTS produces a list of the fonts which are built into Amiga's rom chips. At the present time there are just two of these fonts: Eight-paint Topaz and nine-point Topaz.

=F0NT$ (return details about the available fonts)

a$=FONT$(n)

Returns a string of 38 chars which describes font number n. This function allows you to examine the font list created by a previous call to one of the GET FONT commands.

a* contains a list of characters which hold the name and type of your font. If a font does not exist, a* will be loaded a null value "", otherwise a string will be returned in the following formats

Character	Description
1-29	Font name
30-33	Font height
34-37	Identifier (set to either Disc or Rom)

See EXAMPLE 8.11!

SET FONT (choose a font for use by the TEXT instruction)

SET FONT n

SET FONT changes the character set used by the TEXT command to font number n. If the font is stored on the disc it will be automatically loaded into your Amiga's memory. At the same time any previously sets which are not in use will be removed. See EXAMPLE 8.12.

SET TEXT (set text style)

SET TEXT style

Allows you to change the style of a font. There are three styles to choose from, "style" is a bit pattern in the following formats

Bit	Effect	
0	Underline	By setting the appropriate bits in this
1	Bold	pattern you can choose between a total
2	Italic	of eight different text styles.

=TEXT STYLE (return the current text style)

s-TEXT STYLE

This function returns the text style set from the SET TEXT command. The result in "s" is a bit-map in the same format as that used by SET TEXT.

-TEXT LENGTH (return the length of a section of graphic text)

w=TEXT LENGTH(t*)

The TEXT LENGTH function returns the width in pixels of the character string a* in the current font. The width of a character varies depending on the size of your fonts. In addition, proportional fonts such as Helvetica assign different widths for each individual character.

=TEXT BASE (return the current text base)

b=TEXT BASE

This function returns the position of the baseline of your font. The baseline is the number of pixels between the top of a character and point it will be printed on the screen. It's basically similar to the hot spot of a sprite or bob.

Installing new fonts

If you wish to use your own fonts within AMOS Basic, you'll need to install them onto a copy of your AMOS program disc. The basic procedure is as follows:

- Copy the required font files into the FONTS: directory of your boot-disc.
- Further information can be found in the Extra's manual supplied with the Workbench 1.3 upgrade.

Troubleshooting

Problem: GET FONTS seems to ignore any of the fonts on the current disc.

Solution: You've propably removed the original boot disc from your default drive. The Amiga's library routines expect to find the FONTS: directory on your start-up disc. This can be changed using the ASSIGN program in the UTILITIES folder.

Problem: GET FONTS crashes the Amiga completely.

Solution: This problem can easily occur when you're creating programs in run-only or compiled format. GET FONTS requires the discfont.library in the LIBS folder in order to work.

Problem: The SET FONT command returns a "fonts not examined" error.

Solution: Add a cal to GET FONTS to the start of your program.

AMOS Basic includes a wide variety of the more commonly needed mathematical functions. To conserve memory, AMOS uses the standard Amiga library routines., The appropriate libraries will bs loaded automatically from your workbench disc: the first time you call one of these functions in a particular session., You should therefore ensure that the current disc contains the file MATH!RANS,,LIBRARY in the LIBS folder.

Trigonometric functions

The trigonometric functions provide you with a useful array of mathematical tools. These can be used for a variety of purposes, from education to the creation of complex musical wabeforms.

DEGREE (use degrees)

DEGREE

Generally all angles are specified in radians. Since radians are rather difficult to work with, it's possible to instruct AHflS to accept angles in degrees. Once you've activated this feature any subsequent calls to the trig functions will expect you to use degrees.

RADIAN (use radian measure)

RADIAN

THe RADIAN directive informs AMOS that all future angles are to be entered using radians - this is the default.

=PItt (a constant PI)

att=PI8

This function returns the number called PI which represents the result of the division of the diameter of a circle by the circumference,, PI is used by most of the trigonometric functions to calculate angels. Mote that a $ character is part of the token name! This is to avoid clashes with your own variable names.

= SIW .(sine)

s#=SIN(a)
s#=SIN(a#)

The SIN functions calculates the sine of the angle in n. Mote that the function always returns a floating point number.

=•COS (cosine)

c«=C0S(a|I»3)

The cosine function computes the cosine of an angle.

=TAN (tangent)

t#=TAN(a[#])

TAW generates the tangent of an angle.

=ACOS (arc: cos)

c»=AC0S(n8)

The ACOS function takes a number between -1 and +1 and calculates the angle which would be needed to generate this value with COS.

Note, we haven't provided you with ASIM, because it's not really needed, It can be readily com put (3d using the formula:

ASIN(X)=9O-ACQS(X5 s Rem Measured in degrees.
ASIN(X) = 1.5703-AC0S(X) i Re«i using radians

=ATAN (arc: tangent)

tH=ATAN(ntt)

ATAN returns the arctan of a number.

=HSIN (hyperbolic sine)

s#=HSIN(a[#])

HSIK! computes the hyperbolic sine of angle a.

=HC0S (hyperbolic cosine)

c«=HC0S(a[tn)

HCOS calculates the hyperbolic cosine of angle a.

=HTAM (hyperbolic tangent.)

t»=HTAN(aL"H3)

HTAN returns the hyperbolic tangent of the angle a.

S ta nd a r d ma t h em at i ca1 func t i ons

=1.08 (logarithm)

rtt=L06(v[»])

LOG returns the logarithm in base 10 (log 10) of the expression in vf.

=EXP (exponential function)

rf?=EXP(ett)

Calculates the exponential of eft,, Example:

```
        PrintExp(1)
( result : 2,71828 )
```

=LN (natural logarithm)

r8=LW(1H)

LM computes the natural of naperian logarithm of ltf.

=SQR (square root)

s#=SQR(v[#])

SQR calculates the square root of a number.

=ABS (absolute value)

r=ABS(v[#])

ABS returns the absolute value of v, taking no account of its sign.

=INT (convert floating point number to *An* integer)

i=INT(v#)

INT rounds a floating point number in **v** down to the nearest whole integer.

=S6N (find the sign of a number)

s=SGN(v[#])

SGW returns a value of representing the sign of a number. There are three possibilities,

```
        -1, if v is negative
         0,, if v is zero
         1, if v is positive
```

Creating randorn sequences
==========================

=RND (random number generation)

RND generates a random integer between 0 and n inclusive,, But if n is less than zero, RND will return the last value it produced,, This can be very useful when debugging one of your programs,,

RANDOMIZE (set the seed of a random number) 116

RANDOMIZE seed

In practice, the numbers produced by the RND function Are not really random. They'ris computed internally using a complex mathematical formula. The starting point for this calculation is taken from a number known as the "seed". This seed is set to a standard value whenever you load AMOS Basic into the computer. So the sequence of numbers generated by RND will be exactly the same every time you run your game!

The RANDOMIZE command allows you to set the seed value directly,, so that the numbers would really look like random every time.

"seed" can be any value you wish. In order to generate a. true random numbers, you need some way of varying the seed from game to game. This can be achieved using the TIMER instructions

```
        Randomize Timer
```

TIMER is a Basic function which returns the amount of time which has elapsed since your Amiga was switched on in the current session,, All timings ars measured in units of a 50th of a second.

Manipulating numbers
====================

=MAX (get the maximum of two values)

r=MAX(x,y)
r#=MAX(x#,y#)
r$=MAX(x$,y$) MAX compares two expressions and returns the largest. These expressions can be composed of numbers or strings of characters, providing you don't try to mix different types of expressions in one instruction.

```
        Print Max(10,4)
( result s 10 )
```

=MIN (return the minimum of two values) 117

r=MIN(x,y)
r#=MIN(x#,y#)
r$=MIN(x$,y$) This works the same way the =MAX does, except returns the minimum value of compared numbers/strings.

SWAP (swap the contents of two variables)

SWAP x,,y
SWAP x«',ytt
SWAP x$,,,y$ Swaps the data between any two variables of the same type.

FIX (set precision of floating point output)

FIX(n)

Changes the way your floating point numbers will be displayed on the screen or printer. There Are four possibilities,,

If G<n<3.6 then n denotes the number of figures to be output after thedecima1point,,
If r·i > 16 the p ri n t out w i11 be propo r ti o n a1 a n d a n y t r a i1in g ze ro s w i11 be removed„
If n<0 Then all floating point numbers will be displayed in exponential format, and the absolute value of n will determine the number of digits after the decimal point.

If n=16 then the format will be returned to normal

```
Fix(-4) : Print PI#
```

DEF FN (create a user-defined function)

DEFFNname[(1ist)j-expression

The DEF FN command lets you create your own user-defined functions within an AMOS Basic program. These can be used to compute commonly needed values quickly and easily™

"nane" is the name of the function you wish to define, "list" is a set of variables separated by commas. Only the type of these variables is significant. When you call your function, any variables you enter with, will be automatically subsituted in the appropriate positions.

"expression" can include any of the standard All OS functions you wish. Like all Basic expressions, it's limited just to a single line of prog.

FKi (call a user--defined function)

FNname[(variab1e1ist)3

FW executes a function defined using DEF FN. Examples

```
Def Fn Asin(X)=?0-Acos(X)
Degree
Print Fn Asin(0,,5)
```

The default screen

Whenever you run an AMOS Basic: program a default screen is created as screen zero. This forms a standard display which will be used for all your normal drawing operations,.

The system defaults to a 16-colour screen with dimensions 32()x200, which can easily be altered from within your program. In addition,, you can also define up to seven further screen with power SCREEN OPEN command.,

Definingascreen

SCREEN OPEN (open a screen)

SCREEN OPEN n, w, h,, nc, mods

SCREEN OPEN opens a screen,, and reserves some memory it,, The new screen will now be used as the destination of all subsequent text and graphical operations in your program.

n is the identification number of the screen which is to be created by this instruction. Possible values range from 0-7. If this screen already exists, it will be totally replaced by your new definition.

w holds the width of the screen in pixels. This is not limited to the physical size of your display. It's perfectly lefal to define extra large screens which may be manupulated using SCREEN OFFSET.

h sets the height of your screen using the same system,, Providing you've enough memory, you can easily create screens which are much larger than the visible screen area. These screens can be used in conjunction with all the normal screen operations. So you can construct your images off-screen,, and scroll them into view with the SCREEN OFFSET command.

nc requests the number of colours required for the new screen. The range of available colours varies from 2 to 64 (EHB). You can also access the Amiga's special HAM mode with a value of 4096.

"mode" allows you to choose the width of the individual points on the screen. The Amiga supports screen widths of either 320 or 640 pixels. You can select the required width by setting "mode" either LOWRES (0) or HIRES ($8000).

Here's a list of the possible screen options along with an indication of the amount of memory they consume.

Colours	Resolution	Memory	Notes			
2	320 x 200	8 k	Paper=0	Pen=l	Crsr=l,	no flash
	640 x 200	16 k	"	"	"	"
4	320 x 200	16 k	Paper=l	Pen=2	Crsr=3,	flash=3
	A, Jl 0 v< i7:0 C>	"T 15 I	"	"	"	"
8	320 x 200	24 k	"	"	"	"
	640 % 200	48 k	"	"	"	"

16	320 x 200	32 k	This is a default screen 0
	640 x 200	64 k	
32	320 x 200	40 k	
64	320 x 200	48 k	Extra Half-Bright mode (EHB)
4096	320 x 200	48 k	Hold and Modify mode (HAH)

Note that the memory sizes in the table only apply to a standard screen. If you create taller of wider screens, the amount of memory is consumed will obviously be considerable greater., Screen zero is equivalent tos .

```
SCREEN OPEN 0,,320,,200,16,,Low res
```

SCREEN CLOSE! (erase a screen)

SCREEN CLOSE n

SCREEN CLOSE deletes screen number n, and frees the memory for use.

AUTO VIEW ON/OFF (control viewing mode)

AUTO VIEW OFF

WHen you open a screen using SCREEN OPEN the new screen is usualyy displayed immediately. This can be very incovenient during the initialisation stages of your programs,,

The AUTO VIEW OFF command provides you with full control over the updating process„ It turns off the automatic display system copletely. You can then update the screen display at a convenient point in your program using the VIEW instruction-

AUTQ VIEW ON activates automatic screen updating.

DEFAULT (reset screen to its default)

DEFAULT

Closes all current open screens 3,nd restores the display back to its original default setting. Example:

```
        Load Iff "A:piQS_.DATAsIFF/Affiospic.IFF%0
        Wait Key
        Defaid
```

VIEW (display the current screen settings)

VIEW

Displays any changes to the current screen settings at the next vertical blank pe>riod. You only have to use **this** command when AUTOVIEW is OFF.

Special screen modes

The colour of *every* point on the screen is determined by a value held in one of the Amiga's 32 colour registers. Each register can be loaded from a selection of 4096 different colours,,

Although 32 colours may seem rather a lot, particularly by ST standards, it wasn't enough for the Amiga's designers. The easiest solution would have been to increase the number of colour registers, but this was quickly ruled out from reasons of cost,,

Instead, they invented two special graphics modes which cleveroly 122
exploited the existing registers to increase the maximum number of colours on the screen.

You've propably encountered these modes already. They're the infamous Extra Half Bright and HAH modes. AMOS Basic provides full support for both HAM and Half Bright modes., Here's a brief explanation.

Extra Half Bright mode (EHB)

Doubles the maximum colours on the screen to a grand total of 64. It works by generating two colours for each of the 32 possible colour registers. • .

The first 32 colours load the colour value directly from one of the registers, Each register contains a va1ue between 0 and 4095 which sets the precise shade of the fina1 co1our.

The second group of colours, with numbers from 32 to 63, take one of the previous registers and divide its contents by two. This produces 32 extra colours which *are* exactly half as bright as the normal colour-registers,,

In order to exploit EHB mode to the full, it's necessary to load the 32 registers with the brightest shades in your palette,, This will automatically generate a list of intermediate tones in colours 32-63. Aside from t

Hold and Modigy mode (HAM)

The Amiga's hardware curren11y I imits you to a maxi*m*um of six bit planes per screen. This allows you to display up to 64 different colours on the screen at once. If you wanted to display a photograph 123
though, you'd require hunderds or even thousands of colours on the screen.......

This was the problem faced by Jay Miner when he was designing the Amiga's display system. His solution was to exploit a trick which has been known by artists for centuries. If a professional aritst had to take *every* conceivable colour on an assignment,, he would be faced with an impossible task. It's therefore common parctice to mix the exact shade on the spot, out of a small set of basic colours. This provides millions of potential shades, without the need to carry several large lorry loads worth of paint. The same technique can also be applied to a computer screen. Instead of specifying each colour individually, you can take an existing colour and modify it slightly. This increases the number of available colours tremendously,, and forms the basis of the Amiga's powerfl Hold And Modify mode.

Each colour value on the Amiga is created from a mixture of the three separate components. These determine the relative strength of the primary colours Red,, Green and Blue in the final colour, Possible intenses range from 0 to 15,

Ham mode splits the'Amiga's colour values into four separate groups:

t Colour registers 0-15:; The first 16 colour take a value directly froiTi a colour register. These colours are treadted just like those on a standard 16 colour screen.

* Red components 16-31s However,, if a point is set to a colour nuinberintherange 16 to 31, the colour value is loaded from the pixel to its immediate left. The Red component of this colour is now replaced with a value from 0 to 15 which is calculated from the formula:

  Intensity-Colour index - 16

t Green components 32-47; Similarly, a colour number from 32 to 47 takes the current shade, and changes the green component,, The intensity of this component is set to a value of colour •- 32n

* Blue components 48-63; These colour numbers grab the colour value from the point on the left of the current pixel, and load a new blue component from your colour number like so:

  Intensity - Colour Index - 48 .'•..•

The colour of a particular point therefore depends on the colours of all the points to the left of it. This allows you to create smooth gradiations of colour which *Are* ideal for flesh tones. However, you can't choose the colour of each point on the screen independently. In practice, it takes a maximum of three pixies to shift from one colour to another.

When the Amiga was first released, Ham initially was regarded as little more than curiosity. Nowadays, the situation is *very* different, with the advent of excellent Ham graphics packages such as Photon Paint,,

AMOS allows you to perform the full range text and graphics operations directly on to a Ham screen,, EXAMPLE 10.1 provides you. with a simple example of how you can generate an entire screen in just a few lines of Basic code.

Another point to consider, is that Ham screens *ar&* manipulated using the normal SCREEN) DISPLAY and SCREEN OFFSET commands. Here are some simple guidelines to their uses

t The first point in each horizontal line should be set to a colour number from 0 to 15. This will serve as the starting colour for all the shades on the current line.

* Don't attempt to su*hj*ect your Ham screens to hori*zo*ntal scrolling., If you try to scroll one of these screens, you'll get colour fringes at the sides of your picture. These are generated by the changes in the starting colours for each line. There are no such restrictions to vertical scrolling.

* Fringing [illegible]
  is to ensure that the border of your zone is drawn using a colour from 0 to 15. This will ensure that your Ham screens will be redrawn

at their new position with their original colours„

Loading a screen

LOAD IFF (load an IFF screen from the disc)

LOAD IFF "filename"[„screen]

Loads an IFF format picture from the disc, "Screen" indicates the number of the screen which is to be loaded with your picture. This screen will be opened automatically for your use?,, if it didn't exist„ Anything already inside your screen will be totally erased.

To load the picture into the present screen, omit the "screen" parameter altogether.

Examples

```
Load Iff "Af!0S...DATA5lFF/Af10SPIC.IFF",i
```

Saving a screen

SAVE IFF (save an IFF scree)

SAVE IFF "filename"[,compression]

Saves the current screen as an IFF picture file on the disc, "compression" is a flag which allows you to choose whether your file will be compacted before it's saved,, A value of one specifies that the standard file compressiong system is to be employed and zero saves the picture as it stands,, As a default all AMOS screens *are* compressed.

SAVE IFF automatically appends a small IFF "chunck" to your picture file. This stores the present screen settings including SCREEN DISPLAY, SCREEN OFFSET and SCREEN HIDE/SHOW. When you load this file back into AMOS Basic it will be returned to exactly its original condition. This extra IFF data will be completely ignored by external graphics packages such as DPaint 3.

Note that it's possible to save double buffered or dual piayfieId screens with this command.

Moving a screen

SCREEN DISPLAY (position *A* screen)

SCREEN DISPLAY n [, x„ y„ w, h]

Once you have defined your screen with SCREEN OPEN, you'll need to position it on your screen. Unlike most other computers, the Amiga is capable of displaying a picture anywhere you like on the TV screen. This can be easily exploited to nr-adut-e amazing "boun,-ing" s"oon effects. With AMOS Basic, it's even possible to perform these animations using interrupts (see AMAL).

Another aplication is to overlay several screens alongside each other. This allows you to create ,your display out of a combination of different screen modes.

"n" indicates the number of the screen to be positioned, "x" and "y" specify the location of the screen in hardware coordinates.

The x coordinates of a screen can range from 0 to 448 and are automatically rounded down to the nearest 16-pixel boundary. Only the positions between 112 and 448 actually visible on your TV though, and you *are* strongly advised to avoid using an x coordinate below 112.

The y coordinates of your screen can range between 0 and 312., The visible range will largely depend on your TV or monitor, but you'll propably find that coordinates between 30 and 300 are satisfactory for the majority of systems.,

At the time of writing, there app'ears to be a minor bug in the Amiga's HAM mode. These pictures cannot be displayed with a Y coordinate of exactly 256. So set your coordinates to intermediate values such as 255 or 257 instead. We're not sure if it's a hardware or software fault yet but it won't restrict you by any means,

"w" holds the width of your screen in pixels. If this is different from the original setting, only a part of your image will be shown, starting from the top left corner of the display. Like the x coordinates, the screen width will be rounded to the nearest 16 pixel boundary.

Similarly, "h" sets the apparent height of the screen. Changing this value will reduce the depth of your image™

Generally SCREEN OPEN will automatically select the display position for you using a standard setting in the AMOS configuration file. If a screen is larger than the display then AMOS sets the screen into overscan,,

SCREEN DISPLAY provides you with a simple way of changing these values from the default,, Any of the parameters x,y,h and w may be omitted as appropriate. The unused values will be automatically assigned to the default settings, and should be separated by commas,,

```
        Screen Display 0,3.12,45.,,  s Rem position the screen at 112,45.
```

When you *Are* positioning your screens, try to ensure that the screen starts at the left of the display and ends towards -the right. This is essential if the Amiga's hardware is to interpret your screen correctly. In practice,, you may need to experiment a little to get the precise effect you want. Fortunately,, the worst that can happen is that you'll get a silly looking display. The Amiga won't crash if you make a mistake,, here *are* some guidelines to help you along;

* Only a single screen can be displayed on each horizontal line. However, you can safely place several screens on top of each other. All will *he* well, providing only one of the screens visible.

# There will always be a one pixel thick "dead zone" between each pair of screens. This is generated by the copper list and is completely unavoidable. The dead zone will be noticeable whenever you move a «pri{ₒ b«,+.w.,,.,,r, ╙╘„™ «K= ;„<».>,,.. A.., .»•, <„xsvₘp1ₘ., I,-y novinj the riinuae pointer f ro*m* t he ed i to r w in dow to the menu 1 in e „ You s hou 1 d see a small black line through your mouse pointer at the border between

the two screens.

SCREEN OFFSET (hardware scrolling)

SCREEN OFFSET n,x,y

The Amiga's display is not just limited to the visible dimensios of your TV screen. There's absolutely nothing stopping you from generating image s w h i c h a r e mu c h 1a r g er t han t he a c t u a1 s cr een„ 11's o bv i ous1y not possible to display such pictures in their entirety,, but. you can easily view a section of your image us i n g the SCREEN 0FFSET command.

¹¹n" is the number of the screen to be displayed,, x,,y measure the offset from the top left hand corner of the screen to the starting point for your display, x and y are specified in units of a single pixel, so there's nothing stopping you from gene?rat ing some delightfully smooth scrolls.

You can also use negative offsets with this instruction, allowing you to display any part of the Amiga's memory on the screen. See EXAUF'LE 10.2 for a full demonstration of this command.

SCREEN CLONE (clone a screen)

SCREEN CLONE n

The SCREEN CLONE command assigns a second version of the current screen to screen number n. This clone uses exactly the same memory *ares,* as the original screen.

Normally, the cloned screen is displayed at the same place as its parent. However it can be manupulated separately using any of the normal screen operations such as SCREEN DISPLAY and SCREEN OFFSET.

Since there's only a ^single* copy of the original screen data in memory, you can't access a clone with the SCREEN command. You'll, get an "illegal screen parameter" error if you rty. Another point to consider is that any colour flash sequences you'ye set up on the original screen will NOT be copied during the cloning operation,, See EXAMPLE 10.3. Notice the use of the WAIT V'BL command. This ensures that the clone is re positioned off-screen and keeps the movements running smoothly,,

I f y o u e x p e r i m e n t w i t h S C R E E N C I… 0 ME,, y o u ' 11 q u i c k 1 y f i n d t h a t t h e r e's a real limit to the amount of movement you can perform without spoiling the effect completely. Even something as trivial as an extra calculation to your movement routine can often introduce an unacceptable delay into your animations.

The screen display can also be adjusted directrly from the AMAl- an imat ion language. This is capable of animating large numbers of screens s m o o t h l y a n d e a s i 1 y,, S e e E X A 11 PL E .1.0.4 f o r a d e m o n s t r a t i o n ,.

DUAL PLAYFIELI) (combine two screens into dual piayfield)

DUAL PLAYFIELI) screen!, screen2

The Amiga's dual playfield mode allows you to display two complete screens simultaneously at the same x and y coordinates. It's almost as if you'd drawn eaxh screen on cellophane and overlayed them on top of each ether. Each screen can be manipulated totally independently,, You can exploit this to produce a smooth parallax effect which is ideal for screen scrollinq games such as Silkworm,,

The two components of a dual play-field *Ars* treated just like any other AMOS screen and can be written to in the normal way. They can even be animated within AHAL or double buffered.

"screenl" and "screen2" refer to screens which have been previously defined with the SCREEN OPEN command. Only certain screen combinations *&re* acceptable. Both screens MUST use the same resolution, as it's illegal to use hi res(mean ing actually liedRes 5 and lowres in the same playfield„

Here is a list of the possibilities

Screen 1	Screen 2	Motes
ttof colours	ftof colours	
2	2	
4	2	
4	4.	
8	4	LowRes only
8	8	LowRes only

Although the colour ranges *Are* predefined,, the sizes of the two screens can be completely different. By creating a background screen which is larger than the foreground you can create a delightfully realistic parallax effect.

The colours of these screens *&rs* all taken from the palette of screen! with colour zero being treated as transparent.

Screen	Colour indexes (from screen 1)
1	0 - 7
2	8 - 15

When you are drawing to the second screen., AMOS Basic will au.tomatica11*y*convertyourco1ourindextotheappropriatenumber before using it. So INK 2 will use colour nine from the first palette.

This conversion process does not apply to the assignment statements such as COLOUR or PALETTE. It's important to remember this when you *&re* changing the colour settings, otherwise your new colours will not be reflected on the actual screen,. Always make "screenl" the current screen before changing your colour assignments,,

There *ars* a couple of important opints which you must be aware of before setting up a dual playfield screens

% The screen offsets for both screens must never be set to zero,
t If you set a dual piayfield screen up and then want to position it with SCREEN OFFSET be sure to specify dual screen 1 not the second.

DUAL PLAYFIELI) is an extremely powerful instruction,, A full cieiiiostration can be found in EXAMPLE 10., 5,,

DUAL PRIORITY (choose order of dual playfiek! screens)

DUAL PRIORITY screen1,screen2

The first screen of a dual playfield is normally displayed directrly over the second. The DUAL PRIORITY command allows you to change this order around so that screen? appears in front if screen1

WARNING! This instruction only changes the order of the display. It has $N0# effect on the screen organization. The first screen in the dual playfield list should therefore still be used for all colour assignments and with SCREEN DISPLAY.,

SCREEN (set current screen)

SCREEN n

The SCREEN command allows you to direct all graphical and text operations to screen number n„

=SCREEM (get the current screen tt)

s=SCREEN

Returns the number of the curren11y active screen.

SCREEN TO,FRONT (moves screen to front of display)

SCREEN TO FRONT [s]

This instruction moves screen *"s"* to the front of the TV display,, If the parameter is omitted,, then the current screen will be used instead,,

Note: if the AUTO VIEW system has been turned off,, you'll need to call the VIEW command before the effect will be visible on the screen,.

SCREEN TO BACK (move screen to back of display)

SCREEN TO BACK L'n.1

SCREEN TO BACK moves a screen to the background of your display. If there is another screen at the same coordinate this will now be displayed in front of the selected screen,,

SCREEN HIDE (temporarily hide a screen)

SCREEN HIDE *Lnl*

Removes a selected screen from view copletely,, This screen can be redisplayed using a call to SCREEN SHOW. If n is omitted., this instruction will hide the current screen.

SCREEN SHOW (restore a screen) - 130

SCREEN SHOW [n]

Screen SHOW returns a screen onto the display after it has been hidden with the SCREEN HIDE command.

=SCREEN HEIGHT (return height of screen)

h=SCREEN HEIGHT [n]

Returns the height of an AMOS screen,, If you don't include the parameter n, the height will be returned for the current screen,,

=SCREEN WIDTH (return the width of screen)

w=SCREEM WIDTH En]

SCREEN WIDTH retrieves the width of either the current screen or screen number n. Examples

```
        Print Screen Width
```

=SCREEN COLOUR (return the number of colours)

c=SCR£EN COLOUR

Returns the maximum numbers of colours in the currently active screen.

=SCIN (returns screen number at a selected position)

s=SCIN(x,y)

Returns the number of screen which is underneath the ^'hardwares- coordinates x,y. If this screen does not exist, then s will be loaded withanegativeva1ue(nu11).

SC1N is normally used in conduction with the X MOUSE and Y MOUSE functions to check whether the mouse cursor has entered a particular screen- Examples

```
        Print Scin(X Mouse, Y Mouse)
```

Defining the screen colours 131
===========================

DEFAULT PALETTE (load screen with standard palette)

DEFAULT PALETTE cl, c2, c:3,, ,.. c6, ,,-> up to 32 colours

This command simplifies the process of opening many screens with the same palette,, It defines a list of colours which will be used for all subsequent screens which you create with the SCREEN OPEN instruction. As usual,, the allowable colour values range from $000 to $FFF.

GET PALETTE (set the palette from a screen)

GET PALETTE n [,mask]

The GET PALETTE instruction copies the colours from screen n and loads them into the current screen,, This can be very useful when you're moving information from one screen to another with SCREEN COPY, as it's usually vital that both the source and destination screens share the same colour settings.

The optional "mask" parameter allows you to load just a selection of the colours,, See GET SPRITE PALETTE for full details of mask.

Clearing the screen

CLS (clear the screen)

CLS erases all or part of the current screen,. There are three possible formats of this command:

```
        CLS
```

Clears the current screen by filling it with colour zero and clears any windows which may have been set up.

```
        CLS col
```

Fills your screen with colour col.

```
        CLS col, x1,y1 to x2,y2
```

Replaces the rectangular region at coordinates x1,y1,, x2,,y2 with a block of colour col,, Col can take any value from 0 to the max,, number of available colours. x1,,y1,, x2,y2 hold the coordinates for top left and bottom right corners of the area to be cleared by this command. Example:

```
        Cls : Circle 100,09,09 : Cls 1,50,50 To 150,150
```

Manipulating the contents of a screen

SCREEN COPY (copy sections of the screen)

```
SCREEN COPY scr1 TO scr2

SCREEN COPY scr1,, x1,y1, x2,y2 TO scr2,x3,y3 [,mode]
```

SCREEN COPY makes, it possible to copy large sections of a screen from one place to another at amazing speed-

"scrl" holds the screen used as the source of your image- This can be either a standard screen number or the number of a logical or physical screen generated using the LOGIC and PHYSIC commands.

"scr2" selects an optional destination screen into which this data will be copied. If it's omitted,, the *Area* will be copied into the current screen,.

xi,yl and x2,,y2 hold the dimension of a rectangular source area, and x3,y3 contain the coordinates of the destination,, There *Are* no limitions to these coordinates whatsoever. Any parts of your image which lie outside the current screen *ares,* will be automatically clipped as appropriate.

The optional "mode" parameter chooses which of the 255 possible blitter modes will be used for your copying operation. These modes determine how your source and destination areas will be combined together on the screen,, The mode is set using a bit-pattern in the following format:

node Bit	Source Hit	Destination sett
A	0	0
5	0	i
6	*i*	**0**
7	i	1

Note that the bottom four bits in the pattern *e,re* not used by this instruction and should always be set to zero.

Each bit in "mode" represents a single combination of bits in the source and destination areas. If a mode bit is set to one, then the associated bit on the screen will also be loaded with a one, otherwise the result will be zero,,

In order to select the correct drawing mode for you application, you simply decide which combinations should result in a one and set the appropriate bits in the "mode" parameter accordingly,,

Supposing you only wanted to set a bit on the screen if both the source and destination bits were the same. You would look the table for the points where your requirement was satisfied. This would produce the following vaue for "mode":

```
210010000
```

If you're not familiar with binary notation, you may find this command a little opaque. Rather than boring you silly with an explanation of binay we'll now provide you with a detailed list of the more common requirements along with the associated bit-maps.

Mode	Effect	Bit-pattern
REPLACE	Replaces the destination with a direct copy of the source image (default),,	2:1.1000000
INVERT	Replaces the destination image by a reversed copy of the source image.	200110000
AND	Combines the source and destination with a logical AND operation.,	210000000
OR	OR's the source with the destination	211100000
XOR	Combines the source and destination *Are A* with an Exclusive OR,	201100000

Technically-minded users should note that SCREEN COPY combines the source and destination using blitter areas B and C and that blitter area A is not used by the system at all.

Scrolling the screen

DEF SCROLL (define a scroll zone)

DEF SCROLL n,xl,yl to x2,y2,dx,dy

Allows you to define up to 16 different scrolling zones. Each of these zones can be associated with a specific scrolling operation which is determined by the variables dx and dy,

n holds the number of the zone and can range from 1 to 16, x1,y1 refer to the coordinates of the top left-hand corner of the area to be scrolled and x2,y2 to the point diagonally opposits.

dx signifies the number of pixels the zone will be shifted to the right in each operation. Negative numbers indicate that, the scrolling will be from right to left, and positive numbers from left to right.

Similarly, dy holds the number of pixels the ione will be advanced up or down during the scroll. In this case negative values of dy are used to indicate an upward movement and positive values a downward motion.,

SCROLL (scroll the screen)

SCROLL n

The SCROLL command scrolls the screen using the settings you have specified with the DEF SCROLL instruction, n refers to the number of the zone you wish to scroll.

```
        Load Iff "AMOS_DATA:IFF/Frog_Leap.IFF",2
        Def Scroll 1,0,0, to 320,200,1,0
        Do
          Scroll 1
        Loop
```

Larger examples can be found in EXAMPLE 10.7 and EXAMPLE 10.8,, The variable s holds the number of points the picture will be moved during each SCROLL. Mote the use of screen switching to improve the quality of the motion,,

Screen switching

In order to produce the smooth movement effects found in a computer game, it's necessery to complete all the drawing operations within a time span of no more than a 15th of a second. This represents a real challenge for the fastest computer,, and it's often impossible to achieve erven on the Amiga. If the animation is complex, your graphics will therefore tend to flicker annoyingly as they are being drawn.

Fortunately,, there's a solution at hand which has been succesfully exploited in the vast majority of modern arcade games. This ^screen switching* technique can easily generate flicker-free screen animation using just a fraction of Amiga's computing power,.

Thefaasicideaisextreme1ysiinp1e„Insteadofconstructingyour images on the actual screen, you perform all your drawing operations on a separate logical screen, which is copletely invisible to the user,, This is distinct from the tphysical screen* which is currently being displayed on your TV. 0n ce t he g r aphi cs have been complated ,, you can then swap the logical and physical screen to produce a smooth transition between the two screen imagesn The old physical screen now becomes the new logical screen, and is used to construct the next picture in your sequence.,

At fist glance, this process looks pretty complicated, but it's all performed automatically by the AMOS Basic: DOUBLE: SUFFER''command„ This •forces all drawing operations to be performed directly on the logical screen without affecting the current display. All you need to do within your program is to synchronise your drawing operations with the screen switches. This can be achieved with the help of SCREEN SWAP instruction.

SCREEN SWAP (swap the logical and physical screens)

SCREEN SWAP [n]

SCREEN SWAP swaps the physical and logical screens,, This enables you to instananeously switch the physical display between the two screens,,

If you're using DOUBLE BUFFER,, these screens will have been created for you already. However, you will need to switch off the automatic screen switching system with BOB UPDATE OFF,, as otherwise the screens will be swapped 50 times a second., and will interfere with your own drawing operations. It's also necessary to kill the autoback feature with AUTOBACK OFF. This normally copies your graphical operatains onto both physical and logical screens. It's useful when you wish to combine simple graphics with moving bobs,, but it destroys the effect of your screen switching operations totally.

As an illustration of the power of this command,, have a look at the programs EXAMPLE 10.9 and EXAMPLE 10.10.

=-L0GBASE (return the address of part of part of the logical screen)

address=LQGBASE(plane)

The L06BASE function is aimed at expert programmers who wish to access the Amiga's screen memory directly,, "plane" referes one of the six possible bit-planes which make up the current screen. After LOGBASE has been called, "address" will contain either the address of the required bit-plane, or zero if it doesn't exist.

thecurrentscreen)

address=PHYBASF:

PHYBASE returns the address in memory of bit-plane number "plane" for the current screen. If this plane does not exist, then a value of zero will be returned by this function,. Example:

```
        Loke F'hybase(O) ,0 s Rem pokes a thin line directly onto the
                                  screen„
```

=PHYSIC (return identifier of the physical screen)

=PHYSIC
=PHYSIC(s)

The PHYSIC function returns an identification number for the current physical screen. This number a11ows you to direc11y access the physica1 image which is being displayed by the double buffering system.

The result of this function can be substituted for the screen number in the ZOOM, APPEAR and SCREEN COPY commands.

"s" is the number of an AMOS screen. If it's omitted,, then the present screen will be used instead. Don WOT confuse with the LOGBASE function.

=L0GIC (return identifier of the logical screen)

=L0GIC
=L0GIC(s)

Returns an identification number of a logical screen., This can be used in conjunction with the SCREEN! COPY, APPEAR and ZOOM commands to change your image off-screen, without affecting the current display.

Screen synchronisation

Like most home computers the AMIGA uses a memory-mapped display,, This is a technical term for a concept you *Are* almost certainly already . familiar with,, Put simply, a memory-mapped display is one which uses special hardware to convert en image stored in memory into a signal which can be displayed to your TV screen,, Whenever AMOS Basic accesses the scren it does so through the medium of this screen memory.,

The screen display is updated by the hardware every 50th of a second. Once a screen has been drawn, the electron beam turns off and returns to the top left of the screen,. This process is called the vertical blank period VBL. At the same time, AMOS Basic performs a number of important tasks, such as moving the sprites and switching the physical screen address if it has changed. The actions of instructions such as ANIM or SCREEN SWAP will therefore only be fully completed when the screen is redrawn,.

Since a 50th of a second is a quite long time for AMOS E<asic., this can lead to a serious lack of coordination between your program and the screen, which is especially notice-able in tight program loops. The best way of avoidino this is dif-ficuHf, A« +<> w,n».-t until ths screen has been updated before you. execute the next Basic, command.

WAIT VBL (wait for a vertical blank)

The WAIT WBL instruction halts the AMIGA until ne next vertical talank period. It is commonly used after either a PUT BOB insturction or a SCREEN SWAP

Special effects

APPEAR (fade between two pictures)

APPEAR source TO destination, effect [,pixels]

The APPEAR command enables you to produce fancy fades between the "source" and "destination" screens. Source and destination are simply the numbers of screens you've previously opened using SCREEN OPEN. You can also substitute the LOGIC and PHYSIC functions in these positions if required.

"effect" determines the type of fade which will be produced by this insturction,, The size of this parameter can *vary* from i to the number of pixels in you current screen,,

"pixels" specifies the number of points which *&re* to affected. Normally this value is set to the TOTAL screen area, but you can reduce it to fade only a part of the screen. All screens are drawn in strict order from the top of the screen to the bottom,.

The appearance of your fades will naturally vary depending on the screen mode you *Are* using,, A program is provided in EXAMPLE .1.0.11 to allow you to experiment with the various possibilities.

FADE (blend one or more colours to new colour values) 13?

FADE speed [„colour list]
FADE speed TO screen [,,mask]

The FADE command allows you to smoothly change the entire palette from one set of colours to another. This can be used to generate professional-looking fade effects for your loading screens.

The standard version of the instruction takes the current palette,, and slowly dissolves the screen colours to zero. Each colour value is successively reduced by one until they reach zero,, Example;

```
Fade 15 s Wait 225
```

"speed" is the number of vertical blank periods that must occur before the next colour change is performed,,

Since the fadig effects are executed using interrupts;, it's best to wait until the operation has completely finished before proceeding to the nexy Basic instruction. The time taken for the fade WAIT can be calculated by ths formula:

```
wait value = fade speed * 15
```

Fade c.Afi be extended to generate a new palette directly from a list of colour values.

```
Fade 15,$100,$200,$200,$300
```

Any number of colours can be specified in this instruction,, up to the maximtiHi allowed in the current graphics mode., Like most AMOS commands, it's possible to omit selected parameters completely,, These colours will be totally unaffected fy the FADE command.

```
Fade 15,,,,*100,$800,$F00
```

The most powerful form of FADE smoothly transforms the colours from the current screen into a palette taken from an existing screen.

```
Fade speed TO s C,mask]
```

The present colours are slowly converted into the palette of screen s. It's also possible to load the palette from the sprite bank using the same technique. Simply use a negative value for the screen number s.

"mask" is a bit-pattern which specifies which colours should be loaded. Each colour is associated with a single bit in this pattern numbered from 0 to 15,, If a bit is set to 1, then the relevant colour- will be changed. See EXAMPLE 10.3.2.

FLASH (set flashing colour sequence) 138

This command gives you the ability to periodically change the colour assigned to any colour index., It does this with an interrupt similar to that used by the sprite and the music instructions. The format of the flash instruction is;

```
FLASH index,"(colour,delay)(colour,.delay)(colour,delay),,,,."
```

"index" is the number of the colour which is to be animated. Delay is set in units of a 50th of a second.

Colour is stored in the standard RGB format (See COLOUR) for mode details. The action of FLASH is to take each new colour from the list in turn, and then load it into the index for a length of time specified by the delay. When the end of this list is reached, the entire sequence of colours is repeated from the start., Note that you *are* only allows to use a max. of 16 colour changes in any one FLASH instruction,, Here is a small examples

```
Flash l.,"(007,10)(000,10)"
```

This alternates colour number 1 between blue and black *every* 10/50th of a second»

```
FLASH OFF
```

Turns off the flashing. Note that on start-up, colour number 3 is automatically assigned a flash sequence for use by the cursor,, It's a good idea to turn this off before loading any pictures from the disc.

SHIFT UP (colour rotation)

SHIFT UP delay,,first, last,flag

The SHIFT UP command rotates the values held in the colour registers 'from the "first" to "last". The "first" colour in the list is copied into the second,, and the second into the third,, and so on, until the "last" colour in the series is reached,,

Each AMOS screen can have its own unique set of colour animations. Colour shifts *can* be used to create amazing hyperspace sequences similar to those found in Captain Blood and Elite. Since these animations *are* performed using interrupts, they can be executed while your program is running, without affecting it in the slightest.

"delay" is the time interval between each stage of the rotation, measured in SOths of a second.

"flag" controls the type of rotation,, If it's ste to one,, the last colour index in the list will be copied into the first, and the first • to the last. So the colours will rotate continuously on the screen. When "flag" is set to zero, the contents of the first and last indexes will be discarded, and the region between first and last will be replaced by a copy of the first colour in the list,, For examples

```
SHIFT UP 100,1,15,1

SHIFT UP .10,, 1,15,0
```

SHIFT DOWN (colour rotation)

This is similar to the SHIFT *UP,,* except it rotates the colours in the opposite direction.

SHIFT OFF (stops col,, rotation for the current screen)

SHIFT OFF

Immediately terminates all colour rotations produced by the SHIFT UP or SHIFT DOWN instructions

SET RAINBOW (define a rainbow effect)

Defines an attractive rainbow affect which can be subsequently displayed using the RAINBOW command. It works by changing the shade of a colour according to a series of simple rules.

"n" is the number of your rainbow,, Possible values range from 0 to 3. "colour" is a colour index which will be changed by the instruction,. This colour can be assigned a different value for each horizonal sreen line (or scan line). Mote that only colours 0-15 can be manipulated using this system.

"length'^{1} sets the size of table to store your colours. There's one entry in this table for each colour value on the screen. The size of this table can range from 16 to 54400,, If "length" is less than the physicalHcsigh-tof*yo**trr-«in.nb *T.t*w«:. hc?nlhe? *c:* o1Ourp-tt.t.<? 1"OWX1X*U*t? repeated several times on the screen.

The r$,,g$,b$ command strings, progressibely change the intensities of the red., green and blue components of your final colour,, These values *are* loaded into a special colour table., Each colour in the table determines the appearance of a single horizontal scan line on the screen.

At the start of the rainbow, all the com portents in your colour *3. re* initially loaded with a value of zero. This will be changed according to the information held in the colour table.

Any command string may be omitted if required,, but you'll still have to include the quotes and the commas in their expected positions.

Each string can contain a whole list of commands. These will be cycled continually to produce the final rainbow pattern,, The format is:

```
        (n..step,count) .......
```

"n" sets the number of lintes to be assigned to a' specific colour value in the rainbow. Increasing this number will change the height of each individual rainbow line.,

"step" holds a number to be added to the component. This number will be used to generate the colour of the succeeding line on the screen,, A positive step will increase the intensity of colour component,, and a negative value will reduce it.

Whenever a particular component exceeds the maximum of 15, a new va 1 ue wi 11 be ca 1 cu 1 a ted f rom t he formu 1 a:

```
        new component = old component Mod 15
```

"count" is the number of times the current operation is to be repeated. The best way to demonstrate this command is with an examples

```
        Set Rainbow 0,1.,64,,"(3,2,8)",,"",'"'
        Rainbow 0,,56,1,,255   ..'/,'
        Wait Key
```

This creates a new rainbow with number zero using colour index one. As you can see, SET RAINBOW only defines your rainbow. In order to display it on the screen you need to make use of the RAINBOW command.

The rainbow effects first loads your colour with a value of zero., Everyfourscan-1ines,,theredcomponentwi11beautomatica11y incremented by two. So the contents of colour zero will progressively change from $000 to $EQ0« WHen the component exceeds the maximum of 15, its remainder will be calculated, and the colour will be returned to its starting point (zero). The pattern will now be repeated down the screen„

By defining a separate pattern for eaxh of the red,.green and blue components of your colour, you can easily generate some starling patterns on the screen. Since each rainbow only uses a single colour index, there's nothing stopping you from creating the same effects using just two colour screens. These *Are* ideal from the backgrounds of an arcade game, as they consume very little memory. Example:

```
        Screen Open 0,320,256,2,Lowres
        Set Rainbow 0,1,128,"8,1,8)","(8,1,8)".,""
        Rainbow 0.1,30,128
        Colour 1,0 ; Curs Off : Cls 1 : Flash Off
        Locate 0,2 s Centre "Amos Basic" : Wait Key
```

For further demonstration of the superb effects that can be achieved with this instruction load up EXAMPLE 10.13,,

Rainbows can also be animated using a powerful interrupt system. See the section on AHAL for more details.

RAINBOW-(create a rainbow effect)

RAINBOW n,base,>\,h

Displays rainbow number n on the screen,, If AUTOVIEW is set to OFF,, the rainbow will only *Appear* when you next call the VIEW command..

"base" is an offset in the first colour in the table you created with SET RAINBOW, Changing this value will cycle the rainbow on the screen.

y holds the vertical position of the rainbow in hardware coordinates. The minimum calue for this coordinate is 40,, If you attempt to use a coordinate below this point, the rainbow will be displayed from line 30 onwards,,

h sets the height of your rainbow scan lines.

Rainbows *s.re* totally compatible with the AMOS system including bobs and sprits. However, don't attempt to rainbow a colour which is currently being changed using the FLASH or SHIFT instructions, as this will lead to unpredictable screen effects.

Note that only a single rainbow effect can be displayed on a particular scan line, even if they use different colours on the screen.

Normally the rainbow with the highest screen position will be displayed first. But if several rainbows start from the same scan line,, then the rainbow with the lowest identification number will be drawn in front of the others..

=RAIN (change the colour of an
individual rainbow line)

RAIN(n, line)~c
c=RAIN(n,line)

This is the most powerful of all the rainbow creation commands, as it allows to change the colour of an individual rainbow line to any value you like.

n is the number of the rainbow you wish to access, "line" is the individual scan line to be changed,, Examples

```
Curs Off s Centre "Securitate Stinks!"
Set Rainbow 1,1,409?,"","",""
For Y^O To 4095'
  Rain(.1.;,Y)=Y
Next Y
For 0=0 to 4095--255
  Rainbow 1,0/40,255.
hie x * C
Wait Key
```

ZOOM source,, xl ,yl5x2,y2 TO clest, x3,y3, x4,y4

ZOOM is a simple instruction which allows you to change the size of any rectangular region of the screen..

"source" is the number of a screen from which your picture will be taken. You can also use the LOGIC function to grab your image from the appropriate logical screen. The rectangular *a.reA* to be affected by this instruction is entered using the coordinates xl,yi,x2,y2. "dest" holds the destination screen for your image.. Like the source, it can he either a screen number, or a logical screen specified using LOGIC,

The dimensios of this screen *are* taken from the cordinat.es x3,y3 and x4,,y4,, These hold the dimensios of the rectangle into which the screen segment will be compressed.

The effect of this instruction depends on the relative sizes of the source and destination rectanges.. The source image is automatically resized to fit exactly into the destination rectangle. So the same instruction can be used to reduce or enlarge your images as required.

See EXAMPLE 10.14 for a further demonstration.

Changing the copper list

Ths Amiga's co-processor (copper) provides total control over the appearance of every line on your screen. This copper is a separate processor with its own internal memory and unique set of instructions. By programming the copper it's possible to freely generate a massive variety of different screen effects. Normally the copper is managed automatically by the AMOS system. Each of the available copper effects can be performed directly from within AMOS Basic without the need to indulge in complicated machine-level programming. In practive these intructions will be more*than sufficient for the vast majority of applications.

Obviously, no one can think of everything though. Expert programmers may wish to access the copper directly to create their own special screen modes,,

Be warned! The copper list is notoriously difficult to prograiii, and if you don't know precisely what you *are* doing, you'll almost certainly crash your Amiga. Before embarking on your copper experiments for the first time, you *are* therefore adviced to read one of the many reference books on the subject. A good explanation can be found the "Amiga System Programmers Quite" from Abacus.

COPPER OFF (turn of the standard copper list) 143

COPPER OFF

Freezes the current AMOS copper list and turns off the screen display copletely. You can now create your own display using a series of COP MOUE: A -, dCaPUAiTi.-, ™i. K- … <= -t A „r.».

As a default, all user-defined copper lists *&re* limited to a maximum of 12k. On average,, each copper instruction takes up two bytes. So there's a space for around 6000 instructions,, This may be increased if required, using a special option from the CONFIG utility.

Note that all copper instructions *Are* written to a separate logical list which is not displayed on the screen. This stops your program corrupting the display while the copper list is being created.. To activate your new screen, you'll need to swap the physical and logical lists around with the COP SWAP command.

It's also important to generate your copper lists in strict order,, starting from the top left of your screen and progressing downward to the bottom right. See EXAMPLE: 10.15,

COPPER ON (restart the copper list)

COPPER ON

Restarts the AlʼiOS copper list calculations and displays the current AMOS screens.

COP HOVE (write a MOVE instruction into the logical copper list)

COP MOVE addr,value

Generates a MOVE instruction in the logical copper list.

"addr" is an address of a 16 bit register to be changed. This must lie within the normal copper DATA ZONE ($7F-*1BE). "value" is a word-sized integer to be loaded into the requested register.

COP HOVEL (write a long MOVE instruction into copper list)

COP HOVEL addr,value

This is identical to the COP MOVE,, except that "addr" now refers to a 32-bit copper register, "value" contains a long word intereger.

COP WAIT (copper WAIT instruction)

COP WAIT x,y [,x mask, y mask]

COP WAIT writes a WAIT instruction into your copper list. The copper waits until the hardware coordinates x,y have been reached and returns control to the main processor. Note that line 255 is automatically managed by AMOS. So you don't have to worry about it at all.

x mask and y mask *are* bit maps which allow you to wait until just a certain combination of bits in the screen coordinates have been set. As a default both masks are automatically assignet to tiFF.

COP RESET

Restores the address used by the next copper instruction to the start of the copper list.

COP LOGIC (address of copper list)

addr=CGP LOGIC

This function returns the absolute address in memory of the logical copper list. This allows you to poke your COPPER instructions directly into the buffer, possibly using assembly language,,

Hints and tips

# Before creating a screen with a user defined copper list,, you'll first need to allocate some *memory* for the appropriate bit-maps. Although you can use RESERVE for this purpose, it's much easier to define a dummy screen with the SCREEN OPEN command instead,, The copper registers can be loaded with the addresses of the required bit--maps using the LOGBASE function.

You'll now be able to access your screen using all the standard AMOS drawing features., In order to reserve the correct amount of memory, set the number of colours to the MAXIMUM used in the new screen,, This may be a little wasteful, but simplifies things enormously,

* It's perfectly acceptable to combine user-defined screens with AMOS bobs. If you're using double buffering though, you'll have to define a separate copper list for both the logical and physical screens. This may be achieved using the following procedure?

1 Define your copper list for the first screen
2 Swap the logical and physical copper lists with COP SWAP
3 Swap the physical and logical screens with SCREEN SWAP
4 Define your copper list for the second screen

This will ensure that your bobs will updated correctly on your new screens. All the norma1 A110S commands can be used inc1udina AMA1....

11; HARDWARE SPRUES

One of the biggest attractions of the Commodore Amiga is its ability to produce high quality games which rivial those found on genuine arcade machines. This can be amply demonstrated by terrific programs such as Ba111e Squad ron and E1i m i nator „

Now, for the first time., all these amazing features <nre at your f i n g e r t i p s ! AMOS Basic prov i d es y ou w i t h c omp1e t e c o nt ro1 ove r t he Amiga's hardware and software sprites,, These sprites can be effortlessly manoeuvred with the built-in AMAL animation language,, so you don't have to be a machine code wizard in order to create your own stunning arcade games,,

Hardware sprites *are* searate images which can be automatically overlayed on the Amiga's screen,. The classic: example of a hardware sprite is the mouse pointer,, This is completely independent of the screen, and works equally well in ail the Amiga's graphics modes.

Since sprites don't interfere with the screen background, they *are* perfect for the moving objects required by an arcade game. Not only are they blindingly fast, but they also take up very little memory,, So when you're writing an arcade game, hardware sprites should always be at the top of your list* -

Each sprite is 16 pixews wise and up to 255 pixels high,, The Amiga's hardware supports a maximum of eight three-colour sprites or four fifteen-colour sprites. Colour number zero is transparent - that's the reason for the odd colour ranges,,

At first glance? these features don't seem particulary impressive,, But there *are* a couple of useful tricks which can increase both the numbers and sizes of these sprites beyond recognition,,

One solution is to take each hardware sprite and split it into a number of horizontal segments. These segments can be independently positioned,, allowing you to apparently display dozens of sprites on the screen at once. Similarly, the width restriction can be exceeded by building an object out of several individual sprites. Using this technique it's easy to generate objects up to 128 pixels wide.

Until recently the only way to exploit these techniques was to delve i n t o the mysterious w o1rd of *6* 8000 ass e mb1e r 1a ng u a g e. So y ou ' 11 b e delighted to discover that AMOS Basic manages the entire process automatically! Once you've designed your sprites with the AMOS sprite editor, you can effortlessly manipulate them with just a single Basic: instruction.

The sprite commands

Remember to .have a sprite bank loaded into memory when trying out the various commands in this chapter,, Uie advise you use the file SPRITES.AM from the AI10S data disc.

SPRITE (display a hardware sprits on the screen)

```
SPRITE" r, , x , r , i
```

The SPRITE command displays a hardware sprite on the screen at coordinates x,y using image number i „

n is the identification number of the sprite and can range from 0 to 63. Each sprite can be associated with a separate image from the sprite bank, so the same image can be used for several sprites.

x and *y* hold the position of the sprite using special hardware coordinates. All measurements are taken from the *hot spot* of your images., This serves as a sort of 'handle' on the sprite and is used as a reference point for the coordinates. Normally the hot spot is set to the top left hand corner of an image., However it can be changed within your program using the HOT SPOT command. **146**

Hardware coordinates are independent of the screen mode and effectively start from (••••129,-45) on the default screen. AMOS provides you **with** several built-in functions for conversions between hardware coordinates and the easier to use screen coordinates. See the X HARD, Y HARD, X SCREEN and Y SCREEN functions for more details.

i is the number of a particular image stored in the sprite bank. This bank can be created using the AMOS sprite editor., and is automatically saved along with your Basic program,, It can also be loaded directly with the LOAD instruction. In addition you can use the GET SPRITE command to grab an image straight off the current screen.

Any of these parameters x,y and i *may* be optionally omitted, but the appropriate commas must be included. For example:

```
        Load "AilOSJJATAnSprites/Octopus.abk"
        Sprite 8,200,100,1
        Sprite 8,,.1.50,1
        Sprite 8,300,,
```

For a demonstration of sprites in action, load EXAMPLE 11.1 from the MANUAL folder on the AMOS data disc.

Computed sprites

Although the Amiga only provides you with eight actual sprites, it's possible to use them to display up to 64 different objects on the screen at once. These objects *are* known as -computed sprites-- and *are* managed antirely by AMOS Basic. Computed sprites can be assigned by-calling the SPRITE command with a number greater than 7,, For example,

```
        Load "AMOS...DATAsSprites/Octopus.,abk"
        Sprite 8,2003100,,1
```

The size of a computed sprite is taken directly from the image data, and can *vs.ry* between 16 and 128 pixels wide,, and from 1 to 255 pixels high.

Before you can make full use of these sprites you need to understand soma of the principles behind them. Each hardware sprite consists of a thin narrow strip 16 pixels wide and 256 pixels deep. Depending on the number of colours, you can have either eight or four of these strips on the screen at a time,,

It should be obvious that most of the *area* inside these sprites is effectively wasted. That's because few programs need sprites which are i-iller +. hin about 4 0 or 64 pixels. However there is a simple trick which enables us to borrow this space to generate dozens of extra objects on the screen,, Look at the picture AMOS1..PIC (included in this 147 manual file packet) which contains the letters A^UO and S.

< picture AMOSi.PIC >

This sprite can be split into four horizontal segments each enclosing a single letter. The Amiga's hardware allows each section to be freely-positioned anywhere on the current line, making a total of four computed sprites* Here's a diagram which illustrates this process.

< picture AM0S2.PIC > 148

As you can see, a computed sprite is really just a small part of a hardware sprite displayed at a different horizontal screen position. Notice the line between each object,. This is an unavoidable side effect of the repositioning process, and is generated by the Amiga's hardware.

Due to the way computed sprites are produced, there are a couple of restrictions to their use. Firstly, you can't have more than 8 computed sprites on a single line.. In practice the system is complicated by the need to produce sprites which are larger than the 16 pixel maximum. AMOS generates these objects by automatically positioning several computed sprites side by side., This can be seen from the diagram below:

< picture ANOS3-PIC >

The maximum of eight hardware sprites therefore imposes a strict limit to the number of such objects you can display on a horizontal line,, The total width of the objects must not exceed::

16*8=128 pixels for three-colour sprites 149
16*4=64 pixels for fifteen-colour sprites

If you attempt to ignore limitation, you won't get an error message, but your computer sprite will not be displayed on the screen,, So it's vital to ensure that the above restriction is never broken. This can be achieved using the following procedures

Add together the widths of all your computed sprites., multiplying the dimensios of any fifteen-colour sprites by two., If the total is greater than 128, you'll need to space your sprites on the screen so that their combined width lies below this value„ Take particular care< if you *&re* animating your sprites with A It A I, as certain combinations will only come to light after you've experimented with the sequence for some time. These problems will be manifested by the random disappearance of one or more sprites on the screen..

If the worst comes to the worst., you'll need to substitute some of your larger sprites with Slitter Objects,, This will increase the overall size of your program significantly, but it should have a negligible effect on the final quality of your game.

These restrictions *are* not confined to AMOS Basic of course„ They apply equally well to all games on the Amiga,, even if they're written entirely in machine code! So there's nothing stopping you from producing your own Xenon II clone using exactly the same tehcniques.

Note that, normally,, hardware sprite number zero is allocated to the mouse cursor. You can release this sprite with a simple call to the HIDE command. See EXAMPLE 11.2.

Creating an individual hardware sprite
======================================

The only real problem with computed sprites is that you never know precisely which hardware sprite is going to be used in a particular object. Normally the hardware sprites used in an object will change whenever the object is moved. Occasionally this can be inconvenient, especially when you are animating objects such as missiles which need to remain visible in a wide range of possible sprits combinations.

In these circumstances it's useful to be able to allocate a hardware sprite directly. Individual hardware sprites can be assigned using the SPRITE instruction with an identification number between 0 and 7. Examples

```
        Sprite 1,100.,100,2
```

This loads a hardware sprite number 1 with image number 2. N now corresponds to the number of a single hardware sprite, and can range between 0 and 7. If your image is larger than sixteen pixels wide, AMOS will automatically grab the required sprites in consecutive order starting from the sprite you have chosen,, For examples

```
        Sprite 2,200,100,1
```

Supposing image number 1 contained a 32-bit image with three colours. This command would allocate hardware spries 2 and 3 to the image. Nothing would happen if you were now to attempt to display hardware sprite 3 with a command like SPRITE 3,150,100,1 because this sprite has already been used. You would on 1 *y h*ave access to sprites 0,1,4,5,6 and 7, and the maximum numbers and sites of your computed sprites would be reduced accordingl*y.*

Each 15-colour sprite is implemented using a pair of two three-colour sprites. However,, it's not possible to combinea ny two sprites in this way. Only the combinations 0/1,2/3,4/5,6/7 are allowed. One side effect of this, is that you should always assign your hardware sprites using even sprite numbers. Otherwise, AMOS will start your sprite from the next group of two, effectively wasting the first sprite.

150

Also note that if you try to create a large fifteen-colour sprite with this system, you could easily use up all the available sprites in a single object.

WARNING! If you *are* writing a screen scrolling game, you may encounter problems using sprites in conjunction with the SCREEN OFFSET and SCREEN DISPLAY commands. These generate a DMA clash between the sprite system and the screen bit-maps, and can occasionally lead to unwanted screen effects.

This problem is only relevant if you are using hardware sprites 6/7. When the screen is shifted to the left with SCREEN OFFSET, the amount of time for your sprite updates is reduced, as the screen DMA has priority over the sprite system. Unfortunately, there isn't enough processing time to draw sprites 6/7,, and they will therefore be corrupted on your display.

To clear up this problem, create sprites 6/7 as individual hardware sprites and position them off the screen using negative coordinates. This will stop AMOS Basic from using them in your computed sprites. Providing sprites 6/7 are? never displayed on the screen during your scrolling operations, all will be well.

The sprite palette

The colours required by a hardware sprite *i\re* stored in the colour registers 16 to 31. Providing your current screen (node doesn't make use of these registers,, the sprite colours will be completely separate from your screen colours. Interestingly enough, this is also the case for the 4096-colour Ham {node. So there's nothing stopping you from producing some mind-blowing Ham games with this system!

However you will encounter real problems when using 32 or 64 colour screen in conjunction with three colour sprites. This is because the colours used by these sprites &re grouped together in the following way:

Hardware sprites	Colour registers
0 / j	17 / 18 / 19
2 / 3	21 / 22 / 23
4 / 5	25 / 26 / 27
6 / 7	29 / 30 / 31

Colour registers 16,,20,24 and 28 are treated as transparent,,

The difficulty arises due to the way AMOS generates computed sprites. The hardware sprites used to produce these objects vary during the course of a game, so it's vital to ensure that the three colours used by each individual sprite *are* set to exactly the same values, otherwise the colours of your computed sprites will change unpredictably„ Here's a small AfiOS procedure which will perform the entire process for you automatically,, 151

```
Procedure JNIT._.SPRITES
  Get Sprite Palette
  For 3=0 To 3
    For C==0 To 2
      Colour 3*4+C+17,Colour(C)
    Next C
  Next S
Endproc
```

The above restriction does not, of course, apply to fifteen-colour sprites. If you want to make the most of the Extra Half Bright or 32-co lour modes,, you may find it easier to avoid using four-co lour sprites altogether.

*GET SPRITE PALETTE (grab sprite
colours into screen)

GET SPRITE PALETTE [mask]

This loads the entire colour palette used for your sprite images into the current screen. The optional "mask" allows you to load just a selection of the colours from the sprite palette. Each of the 32 colours is represented by a single bit in the mask, numbered from right to left. The rightmost bit represents the status of colour zero,, the next vit colour 1, and so on. To load a colour simply set the appropriate bit to 1„ If, for instance, you wanted to copy just the first four colours,, you would set the bit pattern tos

```
Get Sprite Palette £0000000000001111
```

Identically, since bobs use the same sprite bank as sprites,, this command can also be used to load the colours of « bob.

Controlling sprites

SET SPRITE BUFFER (set height of the hardware sprites)

SET SPRITE BUFFER n

This sets the work area in which AMOS creates the images of the hardware sprits. Acceptable values for n range from 16 to 256. TO set the correct value for n, simply examine the sprites in the sprite editor and work out which is the largest sprite length wise, ANy sprite that is larger than "n" will simply be truncated at the appropriate cut off point.

SET SPRITE BUFFER is supplied for your use so that you can claim back any redundant memory our game or application simply doesn't use.

The amount of ffiemory consmned by the sprite buffer can be calculated usingtheformu1an

    Memory = W*4£8«3 = W*96

So the miniflurn buffer size is 1336 bytes and the maximum is 24k. Notes This command erases all current sprite assignments and resets the mouse cursor to its original state.

SPRITE OFF (remove one or more sprites from the screen)

SPRITE OFF En]

The SPRITE OFF command removes one or more sprites from the screen. All current sprite movements *Are* aborted. In order to restart them, you'll need to completely reinitialize your movement pattern.

SPRITE OFF   Removes all the sprites from display

SPRITE OFF n Only deactivates sprite number n

Note that your sprites are automatically deactivated whenever you call-up the ADOS Basic editor. They will be automatically returned to their original positions the next time you enter direct mode.

SPRITE UPDATE (control sprite movements)

SPRITE UPDATE [ON/OFF]

The SPRITE UPDATE command provides you with total control of the movements of your sprites. Normally, whenever you move & sprite, its position is updated automatically during the next vertical blank period (see WAIT VBL). But if you *Are* moving a lot of sprites using the SPRITE command, the updates will occur before all the sprites have been moved. This may result in a noticeable jump in yur movement patterns,, In these circumstances, you can turn off the automatic updating system with the SPRITE UPDATE'OFF command,,

One:", *yaiit-* sprites h«v«? bssiwn succcsfully moved, /OH Can thel! 511tlS them smoothly into place with a call to SPRITE UPDATE. This will reposition any sprites which have moved since your last, update,,

=X SPRITE (get *x* coordinate of a sprite)

x=X SPRITE(n)

Returns the current x coordinate of sprite n,, measured the hardware system. This command allows you to quickly check whether a sprite has passed of the edge of the Amiga's screen.

=Y SPRITE (get y coordinate of a sprite)

y=Y SPRITE(n)

Y SPRITE returns a sprite's vertical position. As usual, n refers to the number of the sprite and can range from 0 to 63. Remember, all sprite positions *are* measured in hardware coordinates. See EXAMPLE 11.3

GET SPRITE (load a section of the screen into the sprite bank)

GET SPRITE [s,] i,xl,yi TO *2,,y2

This instruction enables you to grab images directly off the screen and turn them into sprites. The coordinates xl.,yl and x2.,>-2 define a rectangular area to be captured into the sprite bank. Normally all images are taken from the current screen„ However it's also possible to grab the image from a specific screen using the optional screen number "s".

Note; There are no limitations to the region that may be grabbed in this way. Providing your coordinates lie inside the existing screen borders, everything will be fine.

i denotes the number of the new image. If there is no existing sprite with this nuffiber, a new image wi11 be created automatica11y. AMOS w1i1 also take the trouble of reserving the sprite bank if it hasn't been previously defined. See EXAMPLE 1.1.4

There's also an equivalent GET BOB instruction which is identical to GET SPRITE in a\>ery respect.-Since the sprits bank is shared by both bobs and sprites, the images *Are* in exactly the same format,, So it's perfectly acceptable to use both instructions in conjunction with either bobs or sprites,, Try changing the sprite instruction in the previous example to something likes

```
Bob i50,0,1
```

Conversionfunctions

=X SCREEN (convert hardware coordinates
=Y SCREEN into screen coordinates)

x=X SCREENdX,] xcoord)
y=Y SCREEN(Cn.] ycoord)

Transforms a hardware coordinate into a screen corclinate relative to the current screen,, If fhe hardware coordinates lie outside the screen then both functions will return relative offsets from the screens boundaries. Type the following from direct mode;;

```
        Print X Screen (130)
```

The result will be -2. This is because the x screen coordinate 0 is equal to hardware coordinate 128 and thus the conversion of 130 to a screen coordinate results in a position two pixels to the left of the screen.

If the optional screen number is included then the coordinates will be returned relative to screen 8 n„

=X HARD (convert screen coordinates
=Y HARD into hardware coordinates)

X=X HARD (En ,3 xcoord)

These functions convert a screen coordinate into a hardware coordinate. There are four separate conversion functions., the above syntaz converts xcoord from a coordinate relative to the current screen to a hardware coordinate..

Y=Y HARD (En,] ycoord)

Transforms a Y coordinate relative to the current screen into hardware coordinate. As before,, n specif if es a screen number for use with the functions. All coordinates will now be returned relative to this screen.

=1 SPRITE (return current image of a sprite)

Image=I SPRITE(n)

This function returns the current image number being used by sprite n. A value of zero will be reported if the sprite is not displayed.

While hardware sprites are certainly powerful., they do suffer from a couple of annoying restrictions.. The solution is to make use of the Amiga's infamous Blitter chip,, This is capable of copying images to the screen at, rates approaching a mil lion pixels per second! With the help of the blitter it's possible to create what *&re* known as bobs.

Bobs, like sprites,, can be moved around completely independently of > the screen without destoryinq any existing graphics. But unlike sprites, bobs are sroted as part of the current screen,, so you can create them in any graphics mode you wish. This allows you to generate bobs with up to 64 colours. Furthermore the only limit to the number of bobs you can display is dictated by the available memory.

Bobs are slightly slower than sprites and they consume considerably- more memory. Therefore there's a trade-off between the speed of sprites, and the flexibility of bobs. Fortunately there's nothing stopping you from using both bobs and sprites in the same program.

BOB (draw a bob on the current screen)

BOB n, x,, y, i

The BOB command creates bob n at coordinates x,y using the image ft i.

n is the identification number of the bob,, Permissible values normally range from 0 to 63,, but the number of bobs may be increased using an option from the AMOS configuration program,, Providing you've enough memory, you can set this limit to any number you wish.

x and *y* specify the position of the bob using standard screen coordinates. These coordinates are not the same as the hardware coord in a tes used by the equ i va 1 en t SPRITE comrnan d . !…i ke spr i tes,, each bob is controlled through a *hot spot*,, This may be changed at any time with the HOT SPOT command-

i refers to an image which is to be assigned to the bob from the sprite bank. The format of this image is identical to that used by the sprites, so you can use the same images for either sprites or bobs.

After you've created a bob, you can independently change either its position or its appearance by omitting one or more parameters from this instruction. Any of the numbers x,,y or "image" may be left out,, with the missing parameters retaining their original values. This is particularly useful if you *are* animating your bob with AMAL, as it allows you to move your object anywhere you like, without disturbing your existing animation sequence. However you must always include the commas in their original order. Example:

```
        Load  "AMOS_.DATA:Sprites/Octopus.,abk"
        Flash Off s Get Sprite Palette
        Channel 1 To Bob 1
        Bob 1,0,100,1
        Amal i.,"Anim 0,,(1,4)(2,4)(3,4)(4,4)"                    156
        Amal On:
        For X=i; To 320
          Bob  lj, X ? ;
          Wait \Jbl
        Next x
```

Whenever a bob is moved, the area underneath is replaced in its original position,, producing an identical effect to the equivalent SPRITE command.. Unlike STOS on the ST, each object is allocated its own individual storage area. This reduces the amount of memory used by bobs, and improves the overall performance dramatically. Due to the Blitter, of course, therse's no real comparison between STOS sprites and AMOS bobs.

Although the BOB command works fine for small number of bobs,, there's an annoying flicker when you try to use more than three or four objects on the screen at once., This happens because the bobs are updated at the same time as the screen,, You can therefore see the bobs while they *Are* being drawn which results in an unpleasant shimmering effect.

One alternative for improving the quality of your animations is to just limit your bobs to the bottom quarter of the screen,, Since bobs *Are* redrawn extremely quickly, the updates can often be completed before the lower part of the screen has been displayed. This provides you with acceptably smooth movements while consuming *^ery* little memory, so it's a useful trick if you're running short of space. See EXAMPLE 12.1

Obviously this cannot be seen as a serious solution to such a glaring problem. So before you throw away your copy of AH OS Basic: in disgust, you'll be relieved to hear that there's a simple way of eliminating this flicker completely, even when you're using dozens of bobs anywhere on the screen:

DOUBLE BUFFER (create a double screen buffer)

DOUBLE BUFFER

Creates a second invisible copy of the current screen. All graphics operations, including bob movements, *&re* now performed directly in this ^logical screen*, without disturbing your TV picture in the slightest. Once the image has been redrawn,, the logical screen is displayed, and the original ^physical* screen becomes the new logical screen™ The entire process now cycles continuously, producing a rock solid display even when you're moving hundreds of bobs around the screen at once,,

The entire procedure is performed automatically by AMOS Basic,, so after you've executed this instruction you can forget about it completely. Note that since the hardware sprites *are* always displayed using the current physical screen, this system will have absolutely no effect on any existing sprite animations-

Double buffering works equally well in all of the AMIGA'S graphics modes. It can even be used in con j net ion with dual play-fields. But be warned! Double buffering doubles the amount of memory used by your screens. If you attempt to doub1e buf f er too many screens, you ' 11 quickly run out of memory. See EXAMPLE 12.2

In practice, double buffering is an incredibly useful technique, which can be readily exploited for most types of games. It has seen service in the vast majority of commercial games, including Starglider - that's why it's such an integral part of AMOS Basic. A detailed explanation of this process can be found in the SCREENS chapter. AL.so see the SCREEN SUAF- .,,,;,:! AUTOSACK commands.

! SET BOB (set drawing mode of bob)

BET BOB n,back,planes,minterms

The SET BOB command changes the drawing fliode used to display a bob on the screen, n is the number of the bob you wish to affect,

"back" chooses the u&y the background underneath your bob will be redrawn. There are three possibilities:

- A value of 0 indicates that the area underneath your bob should be saved in memory. The old image data is automatically replaced when the bob is moved, resu11ing a smooth movement effect.

- if the "back" parameter is positive then the original background will be discarded altogether;, and the area behind the bob will be filled with colour "back"-!,, This is ideal for moving bobs over a solid block of colour such as a clear blue sky, as it's much faster than the standard drawing system,,

- Turn of the redrawing process completely by loading "back" with a negative value such as -1. You can now deactivate the automatic updating process using BOB UPDATE, and manually move your bobs with a call to BOB DRAW. This allows you to regenerate the screen background using your own customised drawing routines.

"planes" is a bit map which tells AMOS which screen planes your bob will be drawn in. As you. may know, the Amiga's screen is divided up into a number of separate bit-planes. Each plane sets a single bit in the final colour which is displayed on the screen.,

The first plane is represnted by bit one, the second by bit two and so on. Normally the bob is drawn in all the bit-planes in the current screen mode. This corresponds to a bitpattern of "illiiii,

By changing some of these bits to zero, you can omit selected colours from your bobs when they are drawn. This can be used to generate a number of intriguing screen effects,,

"fninterms" selects the blitter mode used to draw your bobs on the screen. A full description of the available modes can be found in the section on SCREEN! COPY, "minterm" is usually set to one of two values;

mi00010	If the bob is used with a mask
*li001010	if NO MASK has been set

Feel free to experiment with the various combinations. There's no danger of crashing your Amiga if you make a mistake. Advanced Amiga users find the following information useful,,

Blitter]source	Purpose
A	Blitter mask
B	Blitter object
C	Destination screen

Note that you afe recommended to use SET BOB fcbefore* displaying your bobs on the screen. If you don't, the Amiga won't crahsh, and you. won't get an error message, but your screen display may be corrputed.

NO MASK (remove blitter mask)

NO MASK [n]

As a default, a blitter mask is automatically created for every bob you display on the screen,. This mask is combined with the screen background to make colour zero transparent. It's also used by the various collision detection commands.

The NO MASK command removes this mask, and forces the entire image to be drawn on the .screen. Any parts of the image in colour zero will now be displayed directly over the existing background.

n is the image number whose mask is to be removed. This mask should never be erased! if the image is active on the screen,, otherwise the sasociated bob will be corrupted. If you must remove the mask in this way, it's important to deactivate the relevant bobs with BOB OFF" first. Here's an examples

```
        Centre "Click mouse button to remove mask"
        Double buffer s Load "AHOSJ)ATA:Sprites/actopus.abk"
        Get Sprite Palette
        Do
          Bob ij,X ScreenCX House),Y ScreensY House),!
          If Mouse Click Then Bob Off s No Mask 1
        Loop
```

See MAKE MASK

AUTOBACK (set automatic screen copying mode)

AUTOBACK n

When you &re using a double hufferend screen, it's essential to synchronize your drawing operations with the movements of your blitter objects. Remember that each double buffered screen consists of two separate displays,, There's one screen for the current picture, and another for the: image whilst it's being constructed. If the background underneath a bob changes while it's being redrawn,, the contents of these screens will be different,, and you'll get an intense and annoying flickering efect.

The unique AMpS AUTOBACK system provides you with a perfect solution to this problem}. It allows you to generate your graphics in any one of three graphics modes, depending on the precise requirements of your program,, Just for a change,, we'll list tese options in reverse order.

AUTOBACK 2 (automatic mode - default)

In this mode,jail drawing operations Are automatically combined with the bob updates. So anything you draw on the screen will be displayed directly underneath your bobs, as if by magic, The principles behing this system c:kn be demonstrated by the following code:

```
        Bob Clear i Rem    Draw on first screen „„. Remove Bobs
        Plot I5y,100 s Rem This can be anything you wish
        Bob Dra^j s Rem    Redraw bobs
        Screen Swap s Rem  Next Screen
        Wait Vb.L
        Bob Clear
        Plot 1501.100 : Rem Perform your operation a second time
        Bob Draw
        Screen Swap s Rem  Get back to first screen
        Wait Vblj
```

As you can see, all screen updates are performed exactly twice. There's one operation for both the logical and the physical screen. See EXAMPLE 12J.3 for a demonstration.

One obvious Iside effect., is that your graphics now take twice as long to be drajwn. Furthermore, the program will be halted by at least 2 vertical blanks, e^ery time you output something to the screen. This may cause: annoying delays in the execution of critical activities such as collision detection.

AUTOBACK 1 (half-automatic: mode)

Performs each graphical operation in both the physical and logical screens. Absolutely no account is taken of your bobs, so you should only use this system for drawing outside the current playing area.

Unlike the standard mode, there's no need to halt your program until the next vertical blank,, Mode 1 is therefore ideal for objects such as control panels or hi-score tables, which need to be upda'ted continually during the game.

AUTOBACK 0 (manda.1 mode)

Stops the AUTOBACK system in it's tracks. All graphics *Are* now output straight to tlje logical screen at the maximum possible speed. You should use this option if you need to repeatedly redraw large sections of your background screen during the course of a game.. This will allow you to safely perform your collision detection routinesatr$gura1intsrva1s,withioutdestroyingtheovera11qua1ity of the animatlon effects. Here's a typical program loop for you to examine.

```
        Bob Update Off
        Repeat  j
        Screen {-Swap
        Wait Vbj
        Bob Cle^r
        Rem Now|redraw any of your gfxs which have changed
        Rem Perform your game routines (Collision detection etc..)
        Bob dratji
        Until WJN
```

Note that this procedure will ONLY work if there's a smooth progression from screen to |cresn. It's entirely up to you to keep the contents of physical and locjical screen in step with each other. An example of this technique can bsi found in EXAMPLE 12.4

Supposing forjinstance, you wanted to display a bob over a series of random blocks, ton might try to use a routine like:

```
        Load "ArTOS_.i}ATA;Sprites/Sprites.abk" : Flash Off
        Get Sprite Palette : Double Buffer s Cls 0 s Autoback 0
        Update Off : Bob 1,160,100,1
        Do
          Bob CleAr
          X=Rnd(320)+l s Y=Rnd(200)+l ; W=Rnd(80)+l
```

```
     H-Rnd(l50]H-l : I=Rnd(i5)
     Ink I !: Bar X,Y To X+W,Y+H
     Rem <tihis would normally call your collision detection routine.:
     Bob I)flaw
     Screed swap : Wait Vbl
   Loop
```

But since there's no relationship between the physical and logical screens, the display will now flick continuously from screen to screen. To overcome this problem, you'll need to mimic the original AUTOBACK system,, Replace the lines in the previous example between the lines Do and Loop as follows:

```
     Rem Update first screen
     Screert Swap : Wait Vbl
     Bob cjear
     X=Rnd|320)+l : Y=Rnd(200)+l ; W=Rnd(80)+l
     H=Rnd(50)+l : I=Rnd(15)
     Ink I! s Bar X,Y To X+W,Y3-H
     Bob Dfaw
     Rem Update second screen
     Screen Swap : UaitVb1
     Bob CJear-
     Ink I : Bar X,Y To X+W,Y+H
     Bob Dfaw
```

The two screens are now updated with exactly the same information,, and the display remains as steady as a rock., even though there's a great deal of activity going on in the background.

Autoback can be safely used at any point in your program. So it's perfectly possible to use separate drawing methods for the different parts of your sjrreen. It's also totally compatible with all graphics operations including Blocks, Icons,, and Windowing.

Bob Control commands

BOB UPDATE (control bob movements

BOB UPDATE [0N/()FF]

Normally all boJDS *are* updated once *every* 50th of a second using a built-int interrupt routine. Alhouth this is convenient for most programs, there are some applications which require much finer control over the redrawing process.

BOB UPDATE OFT turns off the bob updates and deactivates all automatic screen swi tching opera tions if they ' ve been se 1 ectsd . You. may now redraw your bobs at the most appropriate point in your program using the BOB UPDATE command. This is ideal when you *&re* animating a large number of objects as it enables you to move your bobs into position before d rawi ng them on t he screen„ Inev i tab1y this resu11s i n far smoother me-ements in your game.

One word of w<trning: The bob updates will only occur at the NEXT vertical blank, Also note that BOB UPDATE will always redraw the bobs on the current logical screen, so if you forget to use the SCREEN SWAP command, nothing will apparently happen.

BOB CLEAR (remove all the bobs from the screen)

BOB CLEAR

Removes all actj.ve bobs from the screen, and redraws the background regions underneath. It's inteded for use with BOB DRAW to provide an alternative to the standard BOB UPDATE command

BOB DRAW (redraw bobs)

BOB DRAuJ

Whenever the boss &rs redrawn on the screen, the following steps are automatically performed:

1. All active bobs *Are* removed from the LOGICAL screen and the background regions *are* replaced. This step is performed by BOB CLEAR.
2. A list is made of all bobs which have moved since the previous update.
3. The background regions under the new screen coordinates *Are* saved in memory.
4. All active bobs are redrawn at their new positions on the logical screen
5. If the DOUB-E BUFFER feature has been activated,, the physical and logical; screens are now swapped

The BOB DRAW command performs steps 2 to 4 of this process directly,, Supposing you wjished to create a screen scrolling arcade game. In this situation., it would be absolutely vital for your scrolling operations to be perfectly synchronized with movement effects. If the aliens were to move while the scrolling was taking place, their background areas would be redrawn at the wrong place. This would totally corrupt ;/oiir display, and wonId result in a hopeless jumble on the screen. Load EXAMPLE 12.5 for a demonstration of this process.

=X BOB (get X coordinate of bob)

xl=X BOB(n)

Returns the current X coordinate of bob number n. This coordinate is measured relatijvs to the current screen,, See also Y SPRITE,, X 11OUSF and Y HOUSE.

=Y BOB (get Y coordinate of bob)

yl=Y BOB(n)

Y BOB complements the X BOB command by returning the Y coordinate of bob number n. This value will be returned using normal screen coordinates.

=1 BOB (return current image of bob)

Irnage"U BOB(n)

This function returns the current image number being used by bob n. A value of zero w:11 be reported if the bob isn't displayed.

LIMIT BOB (limit a bob to a rectangular region of the screen)

LIMIT BOB [n,] x l,yl TO x2,y2

This command restricts the visibility of your bobs to a rectangular screen area enclosed by the coordinates xl,yt to x2,y2., The x coordinates are rounded up to the nearest 16-pixel boundary. Note that the width of this region must always be greater than the width of your bobs,, otherwise you'll get an "illegal function call" error.

If it's included, n specifies the number of a single bob which is to be affected by this instruction., otherwise *all* bobs will be restricted. You can restore the visibility limit to the entire entire screen by typing:

```
LIMIT BOB
```

GET BOB (load a section of the screen into the sprite bank)

GET BOB [s,] i,xL,yl TO x2,y2

This instruction is identical to the GET SPRITE command. It grabs an image into the sprite bank from the current screen,

xl,yl to x2,yg are the coordinates of the top and bottom corners of the rectangularj area to be grabbed.

i specifies the image number which is to be loaded with this area, s selects an optional screen number from which the image is to be taken. See GET SPRITE "or more details. See also EXAMPLE 12,6.

PUT BOB (fix a x o p y af a bo b o n t o t h e s c r9en)

PUT BOB n

This is the exact opposite of the previous GET BOB command. The action of PUT BOB is ti place a copy of bob number n at its present position on the screen, It works by preventing the background underneath the bob from being redrawnduringthenext.vertica1blankperiod,.Inorderto synchronise the bob updates with the screen display, you should always follow this coffihand with a WAIT VBL instruction.

Note that aftoV this instruction has been performed,, the original bob may be moved or animated with no ill efects.

PAETET BOB (draw an image from the sprite bank on the screen)

PASTE BOB x,y,i

The PASTE BOB cofnmand draws a copy of image number i at *screen# coordinates x,,y. Unlike PUT BOB this image is drawn on the screen immediately, and all the normal clipping rules are obeyed. See PASTE ICON.

BOB OFF (remove a bob from the display)

BOB OFF En]

Occasinoally, you may wish to remove certain bobs from the screen altogether. The BOB OFF command erases bob number n from the screen and terminates any associated animations,, If n is omitted, all bobs will be removed by this instruction.

In this section you will find out how the various objects generated using the sprite and bob commands can be controlled from within an AMOS Basic program, The topics under discussion include collision detection,, using the mouse cursor and reading the joystick.

The mouse pointer

The mouse cursor provides the games programmer with a valuable Alternative to the standard joystick. With the CHANGE MOUSE command you can replace the mouse with an image in the current sprite bank. There's also a group of instructions which allow you to determine both the position and status of this mouse at any time. These include the X HOUSE,, Y NOUSE and MOUSE KEY instructions.

HIBE (remove mouse pointer from the screen)

HIDE COM]

This command removes the mouse pointer from the screen completely. A count of the number of occasions you have called this function is kept internally by the system. This needs to be matched by an equal number of SHOW instructions before the pointer will be returned on the screen,

There's also another version of this instruction which can be >. accessedwithHIDE ON,. This ignores the count and *always* hides the mouse, no matter how many times you've called the SHOW command.

Note that HIDE only makes the mouse pointer invisible. It has no effect on any other AIIOS commands., so you can still use X MOUSE and Y ROUSE functions to read the coordinates of the mouse as normal.

SHOW (activate the mouse pointer)

SHOW [ON]

This returns the mouse pointer to the screen after a HIDE instruction, Works the same way that HIDE does.

CHANGE MOUSE (change the shape of the mouse pointer)

CHANGE HOUSE m

This a11o ws y o u to change the shape of the mouse at any time. Three mouse patterns are provided as standard,. These can be assigned using the numbers 1-3.

If you specify a value m greater than 3, this is assumed to refer to an image stored in the sprite bank. The number of this image is determined using the expression l=m™3. So image number 1 would be installed by a value of 4.

166

In order to use this option, your sprite image must be exactly 16 pixels wide and have no more than four colours. However there's no such limit to the height of your image.

=MOUSE KEY (read status of mouse buttons)

k=MOUSE KEY

Enables you to quickly check whether one or more of the mouse keys have been pressed. It returns a bit-pattern which holds the current status of the mouse buttons.

Bit 0	Set to 1 if the LEFT button pressed, otherwise zero.
Bit 1	Set to 1 if the RIGHT button pressed, otherwise zero.
Bit 2	Set to 1 if the MIDDLE button pressed (if available).

=MOUSE CLICK (check for a mouse click)

c=MOUSE CLICK

Checks wheier this user has "clicked" on a mouse button. Uses the same bit pattern irsdication as ^MOUSE KEY.

One shot tests are only set to 1 when the mouse key has just been pressed. These bits *Are* automatically reset to zero after they've been tested once. So they will only check for a single key press at a time.

=XMOUSE~ (get/set the X coordinate of the mouse pointer)

xl-X MOUSE

X HOUSE returns the current X coordinate of the mouse pointer in hardware notation. You can also use this function to move the mouse on to a specific screen position. This can be achieved by assigning X MOUSE with a value, just like a Basic variable, for example;

X MOUSE=

--YPIOUSE= (get/set the Y coordinate of the mouse pointer)

yi=Y MOUSE

Returns the Y coordinate of the mouse pointer. This can also be used to set the Y position of the mouse pointer the same way as using X MOUSE, See EXAMPLE 13.1 for an example of the X HOUSE and Y MOUSE,.

LIMIT HOUSE (limit mouse to a section of the screen)

LIMIT MOUSE xl.y1 TO x2,,y2

Restricts mouse movements to the rectanjular [illegible] dsfinod [illegible] hardware coordinates (xl,yl) and (x2,y2). Note that unlike LIMIT BOB, the mouse is complstely trapped inside this zone and cannot be moved beyond it. Simply use this instruction with no parameters to restore the mouse to the full screen area.

LIMIT MOUSE

See also EXAMPLE 13.2 from the manual folder for a demonstration,

Reading the joystick

AMOS Basic includes six functions which allow you to immediately check the move merits of a joystick insterted in either of the available sockets.

=JOY(readjoystick)

d=JOY(j)

This function returns a binary number which represnts the current status of a joystick in port number j, Normally your joystick will be placed in the left socket (number 1). However you can remove the mouse from the right-hand socket and replace it with a joystick. This can be accessed using port « 0.

The state of the joystick can be read by inspecting the pattern of binary bits in the resuIt., Each bit indicates whether a specific action has been performed by the user. If a bit is set to one then the test has proved positive and the joystick has been moved in the appropriate direction. Here's a list of the various bits and their meanings!

Bit Number	Significance
0	Joy moved up
1	" down
2	" " left
3	" right
4	Fire button pressed

See EXAMPLE 13.3

You can also use the following commands, if you *are* not familiar with this binary notation:

=JLEFT(j) (test joystick movement left)
=JRIGHT(j) (test joystick movement right)
=JUP(j) (test joystick movement up)
=JDOWN(j) (test joystick movement down)

x=JLEFT(j)
x=JRIGHT(j)
x=JUP(j)
x=JDOWN(j)
	These functions return a value of -I (true) if the joystick in port j has been pulled to the associated direction. Value 0 is reported,, if the condition is false (joystick hasn't been moved to the asked direction).

Detecting collisions

If you're writing an arcade game it's vital to be able to accurately check for collisions between the various objects on the screen,, AMOS

Basic, includes five powerful functions which allow you to perform these tests quickly and easily.

Detecting collisions with a sprite

SPRITE COL. (detect collisions between two hardware sprites)

```
c=SPRITE COL (n [,s TO e])
```

This provides you with a simple &ay of testing to see whether two or more sprites have collided on the screen. The number n refers to an active hardware sprite which is to be checked for a collision. If a collision has occurred a value of -1 (true) will be returned, otherwise the result will be set to 0 (false).

The standard form of this function checks for all collisions. But you can also test a whole group of sprites using an extended version of the command:

```
        c=SPRITE COL n,s TO e
```

The above instruction checks for collisions between sprite n and sprites s to e (inclusive). Once you've detected a collision, you can then get the individual sprite numbers which have vollided using the COL function.

NOTE that in order to use this function, you'll need to create a sprite mask with the MASK command first, otherwise your collisions will not be detected. A detailed example of this command can be found in EXAMPLE 13.4.

Detecting collisions with a bob

BOB COL (detect collisions between two blitter objects)

```
c=BOB(n, [,s TO e])
```

The BOB COL function checks bob number n for a collision with another bob. If a collision has been detected, the value returned in c will be set to -1 (true), otherwise it will be 0.

Normally the command will check for all collisions, but you can specify a collection of bobs to be tested using the optional range parameters s to e. The status of these bobs can be individually examined with the COL command. See EXAMPLE 13.5.

Collisions between bobs and sprites

SPRITEBOB COL (test for a collision between sprites and bobs)

```
c=SPRITEBOB COL(n [,s TO e])
```

This function checks for a collision between SPRITE n ane one or more BOBS. The value of c will be either -1 if a collision has been discovered, or 0 if there have been no collisions. The starting and ending points specify that collisions will only be detected between the bobs s to e. If they *B.re* not included then all active bobs will be tested by this instruction.

WARNING! Collision detection between a sprite and a bob is only possible on a low resolution screen,, In HiRes mode, the pixel sizes used for bobs and sprites *s.re* totally different, and the results from this function will be unreliable,.

BOBSPRITE COL (test for a collision between bobs and sprites)

c=BGBSPRITE COL(n<i,s TO ej)

The BOB SPRITE COL function checks for collisions between a single bob and several sprites. The results and usage of this instruction *&re* same as in the SPRITEBOB COL. See EXAMPLE 13.6.

=COL (test the status of a sprite or bob after a collision detection intruction)

c=C0L(n)

The COL array holds the status of all the objects which have been previously testeri by the collision detection functions,

Each object you have checked is associated with one element in this array,, This element will be loaded with -1 if a collision has been detected with object number n,, or 0 if it has not. The nufiibering system is simples The first element in the array holds the status of object number 1, the second represents object number 2, and so on. See EXAMPLE 13„7•

If you are using the SPRITE COL or B0BSPR1TE COL instructions then the objects will be hardware sprites, otherwise they will be bobs,, In order to avoid confusion, it's sensible to call this instructoin immediatly after the re1evant detection command„

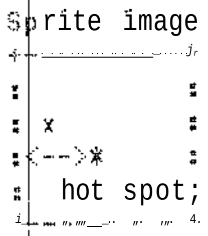
HOT SPOT (set the hot spot for an image in the sprite bank)

HOT SPOT image,x,y
HOT SPOT image,p

This command sets the hot spot of an image stored in the current sprite bank. The hot spot of the object is used as a reference point for all coordinate calculations,, There *Are* two versions of this instruction.

HOT SPOT image,,x,.y

x and y coordinAtes [illegible]
These coordinates will be added to the sprite bank or bob coordinate to position an object precisely on the screen.



Note that it's perfectly iefal for the hot spot to lie outside the actual image,,

HOT SPOT image,. p

This is a short form of the instruction which moves the hot spot to one of nine predefiied positions,, The positions *Are* shown in the diagram below where the centre point of the image is represent by a value of $11.

$00	$10	$20
$01	$11	$21
$02	$12	*22

See EXAMPLE 13.8.

MAKE MASK (make a mask around an image for collision detection)

MAKE MASK [n]

Defines a mask ground image number n in the sprite bank. This is used by all the AMOS Basic collision detection commands. You should therefore creat? *A* mask for every object you wish to check. If you. omit the image number n, then a mask will be generated for each image in the sprite bank. This may take a little time.

It's important to note that masks *&re* generated automatically when a boh is first drawn on the screen. This might cause a significant delay in the running *yf* your program, so it's worthwhile placing an explicit call to MAKE P1AJ3K during your initialisation procedure.

Collisions with rectangu1arb1ocks

AMOS Basic incl.ides a number of functions which allow you to quickly check whether a sprite or bob has entered a rectangular region of the screen„

These screen cones are especially useful for collision detection in rebound games such as Arkanoid as each block can be assignet its own individual sere;n zone. You can also use zones to construct the buttons and switches needed for control panels and dialogue boxes.

RESERWE ZONE (reserve space for a detection zone)

RESERVE ZONE En]i

RESERVE ZONE ali.ocat.es enough memory for exactly n detection zones. This command should always be used before defining a zone with SET ZONE.

The only limij. to the number of zones is the amount of available momavv. so it'^ perfectly feasible to define hundreds or even thousands of zones in one of your programs. To erase the current zone definitions and restore the fflemorybacktothemainprogram,simp1ytype

RESERVE ZONE   with no parameters.

SET ZONE (set a zone for testing)

SET ZONE z,x1,y1 TO x2,y2

Defines a rectangular zone which can be subsequently tested using the various ZONE commands, z specifies the number of the zone to be created and x1,y1 and x2,y2 input the coordinates of the top left and bottom right hand corners of the rectangle..

Before using this instruction you'll need to reserve some space for your zones with RESERVE ZONE.

=ZONE (return the zone under the the requested screen coordinates)

t=ZONE([s],x,y)

ZONE returns the number of the screen zone at the graphic coordinates x,y. Normally the coordinates are relative to the current screen - you can also include an optional screen number s in this function,.

After ZONE has been called, t will hold either the number of the zone at the specified coordinates or a value of 0 (false).,

Note that ZONE only returns the first zone at these coordinates - it won't detect any other zones which lie inside this region.

It is possible to use this function in conjunction with X BOB and Y BOB functions to detect whether a bob has entered a specific screen zone. This can be accomplished using the following codes

```
        X=Zone(X bob(n),Y Bob(n))
```

See Examples 13.9 and 13.10.

=HZONE (return the zone under the requested hardware coordinates)

t=HZONE([s],x,y)

HZONE is almost identical to ZONE except that the screen position is now measured in hardware coordinates. You can therefore use this function to detect when a hardware sprite enters one of your screen zones. For example

```
        X=Hzone(X Sprite(n),Y Sprite(n))
```

See also EXAMPLE 13.11, and ZONE, MOUSE ZONE, SET ZONE and ZONE*

=MOUSE ZONE [illegible] mouse pointer has entered a zone) x=MOUSE ZONE

The HOUSE ZONE •function returns the number of the screen zone currently occupied by the mouse pointer. It's equivalent to the lines

```
        X=Hzone(X mouse,Y mouse)
```

RESET ZONE (erase a zone)

RESET 7.0WE [z]

This command permanently deactivate any of the zones created by SET ZONE. If the optional zone number z is included then only this zone will be reset, otherwise all the zones will be affected. Note that RESET ZONE only erases the zone definitions., it does not return the memory allocated by RESERVE ZONE.

Sob priority

PRIORITY ON/OFF (change between priority modes)

PRIORITY ON/OFF

Each bob is assLgned a priority value ranging from 0-63. Amos basic uses this number to decide which order the objects should be displayed on the screen, As a rule,, any bob with the highest priority will always be displayed in front if any objects with a lower priority. The priority value is taken directly from the number of a Bob,,

You should remember this fact when assigning numbers to your bobs, The choise of number can have wide ranging effects on the appearance of your objects on the screen,

In addition to the standard system, it's also possible to arrange the bobs according to their position on the screen,, PRIORITY ON assigns the greatest priority values to the bobs with the highest Y coordinates, This allows you to create a useful illusion of perspective in your games,, Look at Lhe example below::

```
        Load "AMOSJ)ATA/Sprites/Monkey__right,abk" 2 Cls : Flash Off
        Get Sprite Palette
        Priority Off s Rem Set normal mode
        Bob l,l60,100,2 : Bob 2,0,72,2 : Bob 3,320,128,2 >
        Channel 2 To Bob 2 s Channel 3 to Bob 3
        A dial 2," Loops 11 320,0,320 ; H -320,0,320 ; Jump Loop"
        Amal 3," Loops M ••••320,,0,320 ; Cl 320,,0,320 $ Jump Loop"
        Ama.I On
        Wait Key
        Priority On s Rem Set Y mode
        Wait Key
```

Hormally, both Roving bobs pass below the object in the centre.. When you change the flriority system with a call to PRIORITY ON, the bobs *are* now ranked in order of their increasing Y coordinates. So bob three moves aboce bob one while at the same time, bob two passes smoothly bohind it.

HINT; It's usually best to position the Hot Spot of the sprite at it;

base. This is because the Y coordinates used by this command relate to the position of the Hot Spot on the screen. Also notice that the PRIORITY OFF instruction can be utilised to reset the priority back to normal.

Miscellaneous commands

UPDATE (change automatic sprite/bob updates)

UPDATE [ON/OFT]

Wormal1vanyob,iects you draw on the screen will be automatically redisplayed whenever they *Are* animated or moved. This feature can he temporarily halted using the UPDATE OFF command. When the updates *Are* not active the SPRITE, BOB and A11AL commands apparently have no effect. Actually, all your animations *are* working correctly - it's just that the results *are* not being displayed on the screen,, You can force this redrawing operation at any time using the UPDATE command. Here are the three different forms of the UPDATE instruction.

    UPDATE OFF

Turns of the automatic updating.

    UPDATE

Redraws any sprites which have changed their original positions

    UPDATE

Returns the sprite updating to normal. See EXAMPLE 13,12.

If you wish to generate the smooth movement required in an arcade game, it's necessary io move each object on the screen dozens of times a second., This isj a real struggle even in machine code and it's way beyond the abilities of the fastest version of Basic,

AMOS sidestep•• this problem by incorporating a powerful animation language which is executed independently of your Basic programs, This is capable of gsnerating high speed animation effects which would be impossible in standard Basic,

The (AM)os (A)nimation (L)anguage (AMAL.) is unique to AHOS Basic, In can be used to animate anything from a sprite to an entire scren at incredible speei« Up to 16 AMAL. programs can be executed simultaneously using interrupts.

Each program controls the movements of a single object on the screen, Objects *may* be iiuvci in complex predefined attack patterns, created from a separate editor accessory™ You can also control your objects directly from the mouse or joystick if required,

The sheer versatiiity of the AilAL system has to be seen to be believed.

AMAL principles

ARAL is effect!vely just a simple version of Basic which has been carefully optifinised for the maximum possible speed* As with Basic,, there are instructions for program control (Jump), making decisions (If) and repeatlmg sections of code in loops (For...Next). The real punch comes when AMAL program is run. Mot only are the commands lightning fast but all AHAL programs are *compi led* before run-time,

AMAL commands are entered using short keywords consisting of one or more capital letter-s. Anything in lowercase is ignored completely. This allows you to pad out your AMAL instructions into something more readable., So the M command might be entered as Hove or the L instruction as Let.

AMAL instructions can be separated by parctically any unused characters incl.iding spaces. You can't however,, use the colon ":" for this purpose, as it's needed to define a label. We advise you to use a semi-coIon ";" to separate commands to avoid possible AMAL headaches.

There are two ways of creating your AMAL programs. The first is to produce your animation sequences with the AMAL accessory program and save them into 3. memory bank or you can define your animations inside AHOS Basic usin3 the AMAL command. The general format of this function is:

```
• AMAL n,a$
```

"n" is the identification number of your new AMAL program. As a default all programs *&rf* assigned to the relevant hardware sprite. So the first AMAL program controls sprite number one,, the second sprite number two, and so on. You |:an change this selection at any time using a separate CHANNEL command |. a$ is a string containing a list of AMAL instructions to be performed in your program. Here's a simple example:

```
     Load "AlfiOS_DATAsSprites/!Vionkey__right,,abk"
```

```
Get Sprijte Palette
Sprite 81,130,50,1
Amal 8,"$s M 300,200.,100 •, M ••••300,200,100 J S"
Amal On 8 : Rem Activate AMAL program number eight
Direct
```

The program reti.rns you straight back to direct mode with the DIRECT command. Try typing a few Basic commands at this point. You can see the movement patterr) continues regardless., without interfering with the rest of the AMOS system. Also note we have used sprite 8 to fores the use of a computed sprite. All computed sprites from 8 to 15 *ars* automatica1lya«.signed to the equivalent channel number by the APiAL system,, So ther;'s no need for any special initialisation proceciures. Unless you wish to restrict the amount of hardware sprites it's safest to stick to just computed sprites in your programs. Notice how we've activated the ANAL program using the AMAL ON command. This has the format:

```
AMAL ON [prog]
```

"prog" is the njwnber of a single AMAL program,, If it's omitted,, then *all* your AMAL programs will be executed at once!

AMAL tutorial

We'll now provide you with a guided tour of the AMAL system. This allows you to sjlowly familiarise yourself with the mechanics of AMAL programs, without having to worry about too many technical details,,

For the time seing we'll be concentrating on sprits movements, but the same princiDies can also be applied to bob or screen animations,,

Start off by loading some examples into memory,, These can be found in in the SPRITES folder on the AMOS data disc. To get a directory of Sprite files type the following from the direct windows;

```
Dir "AMJ)S_PATA!"
```

To load a spritj? file, type a line like;

```
Load "AM0S_.DATAsSprites/0ctopus.abk"
```

Moving an object

As you would expect from a dedicated animation language, AMAL allows you. to move your objects in a variety of different ways. The simplest of these involves the use of the Move command.

Move (move object)

M w,h,n

The 11 command moves an object w units to the right and h units down in exactly n movement steps. If the coordinates of your subject were (X,Y)j, then the object would progressively move to X+W,Y+H.

M 100,100,100 would move it to 200,,200. The speed of this motion depends on the number of movement steps. If n is large then each individual sprite movement will be small and the sprite will move very slowly,. Conversely? a small value for n results in a large movement steps which jerk the sprite across the screen at high speed,. Here are some examples of the Move command.

```
        Rem Thi£ moves an octopus down the screen using AMAL
        Load "A!tlOS_pATft:Sprites/Octopus.abk" s Get Sprite Palette
        Sprite £,300,0,1
        Amal 8,til 0,250,50" s Amal On 8 : Wait Key

        Rem Moves octopus down and across the screen
        Load "AIIOSJDATAsSprites/Octopus.abk" s Get Sprite Palette
        Sprite L0,150,150,1
        Afltal 10,"M 300,-100,50" :: Amal On 10 s Wait Key

        Rem Deminstrates multiple Move commands.
        Load "AfiOS_DATA!Sprites/Octopus.abk" 5 Get Sprite Palette
        M$="Mov>? 300,0,50 ; Move -300,0,50"
        Sprite LI,150,150,1
        Amal 11,M$ : Amal On 11 s Wait Key
```

Notice how we'v*; expanded M to Move in above program. Since the letters "ove" *Ar&* in lc*ner* case, they will be ignored by the AMAL system,.

At first glance, Move is a powerful but unexciting little instruction. ItJ's ideal for moving objects such as missiles, but otherwise it's pre11y un :i.nspi r ing „

Actually nothLng could be further from the truth. That's because the parameters in tie move instruction *Are* not limited to simple numbers. You can also use complex arithmetical expressions incorporating one of a variety of us;ful AMAL functions., Example:

```
        Load "AJ10S_,DATAsSprites/0ctopus.abk" : Get Sprite Palette
        Sprite JL2,150,150,1 s Amal 12,"Move XM-X,YM--Y,32"
        Amal On! 12 : Wait Key
```

This smoothly moves computed sprite 12 to the current mouse position. X and Y hold the roordinates of your sprite,, and XM and YM *Are* functions returning the current coordinates of the mouse,,

It's possible to exploit this effect in games like Pac-Man to make? your objects chase the player's character. Examnle:

```
        Load Iff "AM0S_DATA5lFF/Frog._Screen.IFF",l .
        Channel!1 To Screen Display 1
        Amal 1,]'Move 0,-200,50 5 Move 0,200,50"
        Amal On 1 : Direct
```

Channel assigns an AMOS program to a particular object. We'll be discussing this command in detail slightly later, but the basic format is:

```
        CHANNEL:p TO object n
```

"p" is the number of your AMAL. program. Allowable values range from 0 to 63, although;only the first 16 of these programs can be performed using interrupts.

"object" specifies the type of object you with to control with your

```
        Sprite          (values >7 refer to computed sprites)
```

```
Bob     !      (blitter object)
Screen Display (used to move the screen display)
Screen Offfset (Hardware scrolling)
Screen s|ize   (Changes the screen size using interrupts)
Rainbow !      (Animates a rainbow effects
```

"n" is the numbejr of the object to be animated. This object needs to be subsequently **defined** using **the SPRITE,** BOB or SCREEN open instructions.

Animation

Anim (animate an object)

H n „(image,deia>j) (image,delay) „„„ .

The Anim instruction cycles an object through a sequence of images, producing a smooth animation effect,, "n" is the number of times the animation cycle j is to be repeated., A value of zero for this parameter will perform thej' animation continuously.

"image" sprcifies the number of an image to be used for each frame of your animation., j "delay" determines the length of time this image is to be displayed on the screen, measured in units of a 50th af a second.
Examples j

```
Load "AfjlOS_DATA:Sprites/Ptonkey_right-abk" s Get Sprite Palette
Sprite f,150,50,11
M*="Aniij»'i2, (1,4)(2,4)(3,4)(4,4)(5,4)(6,4) ;"
i1$=M$+"lflove 300,,150,,150 ; Rove -300,-150,75"
Amal 9,if!*
Amai On 9
Direct
```

This program combines a sprite movement with an animation. Notice how we've separated!the commands with a semi-colon. This ensures that the two operations Are totally independent of each other. Once the animation sequence has been defined, AHAL will immediatly jump to the next instruction, and the animation will begin.

It's important to realize that Anim only works in conjunction with sprites and bob^. So it's not possible to animate entire screen with this command„

Simple Loops

Jump (redirects an AMAL program)

J label

Jump provides a simple way of moving from one part of an AMAL. program to another, "label" is the target of your jump, and must have been defined elsewhere in your current program. All AilAL labels are defined using a single uppercase followed by a colon., like instructions, you can pad them out with lower case to improve readability.

Remember that each label is deinfed using just a *single* letter. So "Ss" and "Swoops" refer to the same label! If you attempt to define two labels starting with an identical letter, you'll be presented with a "label already defined in animation string" error,,

Each AMAL program can have its own unique set of labels. It's perfectly acceptable to use the identical labels in several different programs. Example: • •

```
Load "AMtj)S_DATAsSprites/Octopus.abk"
Get Sprite Palette
For S~8 -to 20 Step 2 : Rem Set up ? computed sprites
 Sprite £,200,(S-7)*13+40,l
Next S
Rem : Woifj let's create seven AMAL. programs
For S=l to ?
 Channel |S To Sprite 6+(S*2)
 PI$="Anii|t 0,(i,2)<2,2)(3,2)(4,2) ; Label: Move "+Str*(S*2)"+50,7 ;"
 Amal S,lf!$
Next S
Rem OkayL now animate it all!
Amal On e Direct
```

Since AMAL commands are performed using interrupts, infinite lopos could be disastrous. So a special counter is automatically kept of the number of jumpsi in your program,, When the counter exceeds ten, any further jumps will! be totally ignored by the AHA!... system.

NOTE: if you fely on this system, and allow your programs to loop continually, uoj.i'11 waste a great deal of the Amiga's computer power, In practice., itj's much more effecient to limit yourself to just a single jump perj VBL. This can be achieved by adding a simple PAUSE comand before each Jump in your program. See PAUSE for more details.

Variables and expressions

Let (assigns a value to a register)

L register=expression

The L instruction assigns a value to an AMAL register. The action is *very* similar to normal Basic, except that all expressions *&re* evaluated strictly from left to right,,

Registers are integer variables used to hold the intermediate values in your AMAL programs,, Allowable numbers range between ••-32763 to +32768. There are three basic types of register;

Internal registers

Every AMAL program has its own set of 10 internal registers. The names of these registers start with the letter R, followed by one of the digits from 0 to 9 (R0-R9). Internal registers are like the local variables inside an AMOS Basic procedure,,

External registers

ticternal registers are rather different because they retain their values between separate AMAL programs. This allows you to use these registers to pass information between several AMAL routines. AMAL provides you with up to *26* external registers,, with names ranging from RA to RZ. The contents of any internal or external register can be accessed directly from your Basic program using the AHREB function•

Special registers

Special registers &r<s a set of three values which determine the status of youtf object. X,Y contain the coordinates of your object,, By chanaing these; registers you can move your object around on the screen. [Example:

```
Load "AiiOSJ)ATAsSpr:Ues/F>og_j3prites..abk" ". Channel 1 To Bob i
Flash Off : Get Sprite Palette s Bob 1,0,0,1
Amal 1,"Loops Let X=X+1 ș Let Y=Y+l; Pause;; Jump Loop"
Amal On 1 ; Direct
```

"A" stores the number of the image which is displayed by a sprite or bob. You can alter this value to generate your own animation sequences like so:

```
Load "AI10S_DATA:Sprites/Frog_Sprites.abk" : Get Sprite Palette
Flash Off ; Channel 2 To Bob 1 5 Bob 1,300,100,1
i1$="Loop; Let A=A+1 ; "
M$=M$+"for R0=l To 5 ;; Next R0 ; Jump Loop"
Amal 2,!j1$
Afflal On! 2 i Direct
```

Th& For To Next lop will be explained in more detail below. It is used here to slow dojrn each change to Bob l's image. When the "Next" of the loop is execute^, Ail A I. won't continue until a vertical blank has occurred. Also note the use of ";" to separate the AMAL instructions - although a space " " will serve just as well.

Operators

AMAL expressions can include all the normal arithmetic operations, except MOD., You! can also use the following logical operatoins in your calculations:

```
&      ! Logical AND
!      ; Logical OR
```

Note that it's not possible to change the order of evaluation using brackets "()" as this would slow down your calculations considerably and thus reduce the allowable time in the interrupt. Type the following examples

```
Load "A(10S_DATA:Sprites/0ctopus.abk" s Hide
Get Sprite Palette
Sprite 8..X Mouse,Y Mouse, 1
Amal 8,"Loop:: Let X=XM ;' Let Y=YM 5 Pause ; Jump Loop"
Amal On 8

Load "AM0S_DATA:Sprites/0ctopus.abk" s Hide
Get Sprite Palette
Sprite 8,X House,,Y Mouse., 1
Amal 8,"Ani(n 0, (1,4) (2,4) (3.4) (4,4) ; Loops Let X=XM 3 Let
                                          Y=YM 1 Pause ; Jump Loop"
Afnal On;
```

The above examples effectively mimic the CHANGE MOUSE command. However L h i !5 is y t^r t GJ m :i s much more? p o w e r f u X <s » y o 11 c: t\ n e? <A 3 i- X y move b o b s ., <7. <s tn p u t &? d sprites, or even screens using exactly the same technique.

If (branch within an AMAL string)

If test Jump L

This instruction allows you to perform simple tests in your AMAL programs. If thi> expression test is -1 (true) the program will jump to label L, otherwise AMAL will immediately progress to the next instruction. Note that unlike it's equivalent., you're limited to a single jump operation after the test.

It's common practice to pad out this instruction with lowercase commands like "then" or "else". This makes the action of the command rather more obvious. Here's an example;

```
        If X>100 then Jump Label else Let X=X+i
```

"test" c&n be aiiy logical expression you. like, and may include;

<>	Noti equals
<	Lesf; than
>	Greater than
=	Equals

Examples

```
        Load "A!flOS_.I)ATAsSprits/Octopus,,abk"
        Get Sprite Palette
        Sprite $,130,50,1
        C*="Main; If X)1>100 Jump Test:: "
        C*=C*+"llet X=XM "
        C*=C$+"fest: If YMX10Q Jump Main "
       . C*=C*+"Let Y=YM Jump Main"
        Amal 8,C$ ; Amal On : Direct
```

WARNING! Don't try to combine several tests into a single AMAL expression using "&" or *"I".* Since expressions *Are* evaluated from left to right, this mill generate an error. Take the expressions XM00JYM00. This is intended to check whether X>100 OR Y>100. In practice,, the expression will be evaluated in the following order;

```
        X>100   May be TRUE or FALSE
         !Y     :0R result with Y
        >10()   :Check if (Y>100 j Y)>100)
```

The result from the above expression will obviously be no relation to the expected value. Technically-minded users can avoid this problem by using boolean algebra. First assign each test to an single AHA!... register like so:

```
        Let R0=X>100; Let Rl=Y>100
```

Now combine these tests into a single expression using J and &. and use it directly in your If statement.

```
        If R0 ! Ri Jump !...
```

This may look a little crazy., but it works beautifully in practice.

For To Next (loop within AMAL.5

```
For reg=start To end
 :   :
Next req
```

This implements a standard F0R...MF.XT loop which is almost identical to its Basic equivalent. These loops can be exploited in your programs to move objects in complex visual patterns,, "reg" may be any normal AMAL register (R0--R9 or RA-RZ),, However you can't use special registers for this purpose.

As with Basic, the register after the Next must match with the counter you specified in the For,, otherwise you'll get an AMAL syntax error. Also note that the step size is always set to one. Additionally, it's possible to "nest" any number of loops inside each other.

Note that each animation channel will only perform a single loop per VBL. This synchronizes the effects of your loops with the screen display, and avoids the need to add an explicit Pause command before each Next.

Generating an attack wave for a game

These lopes can; be used to create some quite complex movement patterns. The easiest type of motion is in a straight line. This can be generated using a single for... .Next loop like so;

```
••' Load "AhOS_DATA:Sprites/Gctopus.abk" 5 Get Sprite Palette
    Sprite 8,130,60,1
    C$=For R0=l To 320 5 Let X=X+1 ; Next R0" 5 Rem Move sprite
    Amai 8,C$ : Amal On 8 s Direct
```

You can now expand this program to sweep the object back and forth across the screen.

```
Load "AiiOS...SATA?Sprites/Octopus,,abk" : Get Sprite Palette
Sprite 8,130,60,1
C*="Loop: For R0=l To 320 p Let X=X+l 5 Next R0 ;"
C*=C$+"'For R0=.1. To 320 ; Let X=X-1 5 Next R0 ; Jump Loop"
Amal 8,C* : Amal On 8 : Direct
```

The first loop moves the object from left to right, and the second from right to left. So far the pattern has been restricted to just horizontal movements,, In order to create a realistic attack wave, it's necessary to incorporate a vertical component to this motion as well. This can be achieved by enclosing your program with yet another loop.

```
Load "AM0SJDATAsSprites/0ctopus,,abk" s Get Sprite Palette
Sprite 8,130,60,1 : C*=For Rl=0 To 10 ;"
C$=C$+"For R0»l To 320 ; Let X=X+1 ; Next R0 ; "
C$=C$f"Let Y-+Y+8 ; "
C$=C$+"For R0=l To 320 5 Let X=X-1 ; Next R0 ; "
C*=C*+"Let Y=Y+8 ; Next Rl"
Anial 8,C$ : Amal On 8
```

The above programs generates a smooth but quite basic: attack pattern. A further demonstration can be found in EXAMPLE: 14.1 in the MANUAL folder.

Recording a complex movement sequence

PLay path

If you've? looked at the smooth attack waves in a modern arcade game, and thought them forever beyond your reach, think again. The ARAL Play command allows you freely animate your objects through practically any sequence of movements you can imagine,. It works by playing a previously defined movement pattern stored in the AHAL memory bank.

These patterns are created from the A HAL. accessory on the AMOS program disc. This simply records a sequence of mouse movements and enters them directly into the amal memory bank. Once you've created your patterns in this way, you can effortlessly assign them to any object on the screen, reproducing your original patterns perfectly. Both the speed and direction of your movement can be changed at any time from your AMOS Basic program.

The first time AHAL encounters a Play command, it checks the AHAL bank to find the recorded movement you specified using the "path" parameter, "path" is simply a number ranging from one to the maximum number of patterns in the bank. If a problem crops up during this phase, AHAL wi11 abort the p1ay instruction comp1ete1y,, and wi11 skip to the next instruction in your animation string,,

After the pattern has been initialised, register RO will be loaded with the tempo of the movement. This determines the time interval between each individua1 movement step„ A11 timings are measured in units of a 50th of a second. By changing this register within your AMAL. program, you can speed up or slow down your object movements accordingly.

Note that each movement step is Kadded* to the current coordinates of your object. So if &n object is subsequently moved using the Sprite or Bob instructions, it will continue its manoeuvres unaffected, starting from the new screen position. It's therefore possible to animate dozens of different objects on the screen using a single sequence of movements.

Register Rl now contains the flag which seta the direction of your movements. There Are three possible situations:

* R1 > 0 Forward

A value of one for Ri specifies that the movement pattern will be replayed from start to finish,, in exactly the order it was created (this is the default).

* Rl=0 Backward

Many animation sequences require your objects to move back and forth across the screen in a complex pattern,, To change direction, simply load Ri with a zero. Your object will now turn around and execute your original movement steps in reverse.

* Rl=-1 Exit

If a collision has been detected from your AMOS program,, you'll need to stop your object completely, and generate an explosion effect,, This can be accomplished by setting Rl to a value of minus one. AMAL will now abort the play instruction,, and immediately jump to the next **instruction in your** animation sequence.

The clever thing about these registers is that they can be changed directly from AilOS Basic, This lets you control your movement patterns directly from within your main program. There's even a special AMPLAY instruction to make things easier for you.

The PLay comand is perfect for controlling the aliens in an arcade game. In fact, it's the single most powerful instruction in ANAL.

AHAL (call an AHAL program)

ANAL n,a*
AilAL n,p
A HAL. n,a* to address The AMAL command assigns an ARAL program to an animation channel. This program can be taken either from a string in a$ or directly from the AI1AL bank.

The first version of the instruction loads your program from the string a$ and assigns it to channel n. a> can contain any list of AHAL instructions,, Alternatively you can load your program from a memory bank stored in bank number 4.

n is the number of an animation channel ranging from 0 to 63. Each AMOS channel can be independently assigned to either a bob, a sprite or a screen.

Only the first 16 AHAL. programs can be performed using interrupts. In order to exceed this limit you need execute your programs directly from Basic using the SYNCHRO command,,

The final version of the AHAL insturction is provided for advanced users. Instead of moving an actual object,, this simply copies the contents of registers X,Y and A into a specific area of memory. You can now use this information directly in your own Basic routines. It's therefore possible to exploit the AM A I… system to animate anything from a Block to a character. The format is:

    AMAL n,a$ To address

"address" must be EVEN and must point to safe region of memory, preferably in an AMOS string or a memory bank,, Every time your AMAL program is executed (50 times per second), the fo11owing va1ues wi11 be written into this memory area;

Location	Effect
Address	Bit 0 is set to 1 if the X has changed Bit 1 indicates that Y has been altered Bit 2 will be set if the image -(A) has changed since the1astinterrupt„
Address+2	Is a fcword* containing the latest value of >;
Address+4	Holds the current value of Y
Address+6	Stores the value of A

These values can be accessed from your program using & simple DEEK., NOTE; This option totally overrides any previous CHANNEL assignments.

AMAL commands

Here is a full list of the available amal commands!;

H (Move)	Move deltaX, deltaY, steps		
A (Anim)	Ani(n cycles,,(image, delay)(image,.delay)...		188
L (Let)	Let reg=exp		
J (Juflip)	Jump L		
I (If)	If exp Jump L		
For To Next	For Reg=start To end ...Next Reg		
PL (PLay)	PLay path		189
P (Pause)	Pause		
AU (AUtotest)	AU (list of tests)	See the Autotest System	190
X (eXit)	sXit	Exits from an AUtotest and re-enters the current AMAL program.	
U (Wait)	Wait	Freezes your AMAL program and only executes the AUtotest.	
O (On)	On	Activates the main program after a Wait.	
D (Direct)	Direct	Sets the section of the main program to be executed after an autotest.	

AMAL functions 191

=XM	Returns the X coordinate of the mouse
=YM	Returns the Y coordinate of the mouse
=K1	Status of left mouse key (-1, if pressed, otherwise 0)
=K2	Status of right mouse key
=J0	Test right joystick. Result in bit-map.
=J1	Test left joystick. See the JOY command.
=Z(n)	Random number. Returns a random number between -32767 to 32768. This number can be limited to a specific, range using the bit-mask n. A logical AND operation is performed between the bit mask n and the random number to generate the final result. So setting n to a value of 255 will ensure that the numbers will be returned in the range 0 to 255. Since this function has been optimized for speed, the number returned isn't totally random. If you need really random numbers, you would be better to generate your values using Basic's RND and then load them into an external AMAL register with the AMREG function.
=XH(s,x)	Converts a screen x coordinate into a hardware coordinate. 192
=YH(s,y)	Converts a screen y coordinate into hardware format.
=XS(s,x)	Hardware to screen conversion
=YS(s,y)	Hardware to screen conversion
=BC(n,s,e)	Check for collisions between bobs. BC is identical to the equivalent AMOS Basic BOB COL instruction. It checks bob number n for collisions between bobs s to e. If a collision has been detected, then BC will return a value of -1, otherwise 0. This instruction may NOT be performed within an iterrupt. So it's only available when you *are* executing your AMAL routines directly from Basic with the SYNCHRO instruction.
=SC(n,s,e)	This is equivalent to the SPRITE COL. function. Like BC function, it's only allows in conjuction with the SYNCHRO instruction.

=V(v)   VU-meter. The VU function samples on& of the sound channels and returns the intensity of the current voice. This is a number in the range 0-255. You can use this inforfliation to animate your objects in time to the music. An example of this can be found in EXAMPLE .14,.3» Also ses the VUMETER function from AMOS Basic

AilAL ON/OFF' (start/stop an AilAL program)

AilAL ON [n]

Once you've defined your AilAL program you need to execute it using the AHAL ON command. This activates the AMAL system and starts your prografiis from the first instruction,

AilAL OM activates all your programs,, The optional parameter n allows you. start just one routine at a time?.

AilAL OFF [n]

Stops one or all ARAL programs from executing. These programs are erased from meomry. They can only be restarted by redefining them again using the AilAL instruction.

I   AMAL FREEZE (temporarily freeze an amal program)

AilAL FREEZE [n]

Stops one or more AilAL programs for running. Your programs can be restarted at any time using a simple call to AHAL ON. Note that this instruction should always be used to stop AMAL before a command such as DIR is executed, otherwise problems with timing *can* cause visual mishaps.

=A11REG"" (get the value of an external AHAL register)

r=AHGER(n, [channel])
Ai7!REG(n, [channel])=-expression

The AilREG function allows you to access the contents of internal and external AMAL register directly from within your Basic program,,

"r." is the number of the register,, Possible values range from 0 to 25 with zero representing register RA and twenty-five denoting RZ.

By using the optional "channel" parameter you can reference any AilAL internal register. In this mode "n" ranges between 0 and 9 representinq R0 to R9.

The following ©Xamples shows how it is possible to retrieve a sprite's current X-position from Basics

```
        Load »AMOSJ>ATA:Sprites/Octopus.abk" : Get Sprite Palette
        Channel 1 To Sprite 8 : Sprite 8,100,100,,!
        At="Loop: Let RX=X+1; Let X=RX; Pause? Jump Loop"
        Amal 1,A* s Amal On ; Curs Off
        Do
          Locate 0,0
          Z=Asc("X")-65 s Rem Note the use of ASC to get the register
          Print:Amreg(Asc("X")-65)
        Loop :
```

! AIIPLAY (con t ro 1 an an i ma t i on . ; produced with PL ay) ." 194

AII PL. AY tempo ..direction [start TO end]) -

Any movement sequences you've produced using the AI1AL PL. command *are* controlled through the internal registers R0 and Rl. Each object will be assigned it's own unique set of APIAL registers. So if you're animating several objects, you'll often need to load a number of these registers with exactly the same values.

Although this can be achieved using the standard AHREG function, it would obviously be much easier if there was a single instruction which • allowed you to change R0 and Rl for a. whole batch of objects at a time. That's the purpose of the AMPLAY command.

AilPLAY takes the "tempo" and "direction" of your movements, and loads them into the registers R0 and Rl in the selected channels.

"tempo" controls the speed of your object on the screen- It sets a delay (in 50ths of a second) between each successive movement step,

"direction" changes the direction of the motion. Here's a list of the various different options.:

Value	Direction
X)	Move the selected object in the original movement direction.
0	Reverses the motion and moves the object backwards
-1	Aborts movement pattern and jumps to the following instruction in your A HAL animation sequence,.

As a default, this instruction will affect ail current animation channels. This can be changed by adding some explicit "start" and "end" points to the command, "start" is the channel number of the first object to be adjusted., "end" holds the channel number assigned to the last object in your list. Mote that either the "tempo" or the "direction" can be omitted as required. Examples;;

```
        Am pi ay ,0 :; Rem reverse your objects
        Amp 1 ay 2, s R>.?m S1 ow down you r movemen t pa 11erns
        Amplay ,-i 3 To 6 s Rem stop movements on channels 3,4,5 and &.
```

=CHANAN (test AI1AL animation) .195

s=CHANAN(channel)

This is a simple function which checks the status of an AMAL animation sequence and returns -1 (true) if it's currently active or 0 if the animation is complete,, "channel" holds the number of the channel to be tested„

=CHANMV (checks whether an object is still moving)

s=CHANMV(channel)

Returns a value of -1 if the object assigned to "channel" is currently moving, otherwise 0 (false).

This command can be used in conjunction with the AMAL Move instruction to check whether a movement sequence has "run out" of steps. You can now restart the sequence at the new position with an appropriate movement string if required,, Example:

```
Load "AMOSJ>ATA;;Sprites/l1ankey_...right»abk" s Get Sprite Palette
Sprite 9..i50,50,11
M*=flove 300,150,150; Move -300,-150,75"
Amal ?,i'i* s Amal On
While Chanmv(9)
Wend
Print "Movement complete"
```

APIAL errors

=A!1ALERR (return the position of an error)

p=AMALERR

Returns the position in the current animation string where an error has occurred. Careful inspection of this string will allow YOU to quickly correct your mistakes. Examples

```
Load "Ai10S_DATA3Sprites/0ctopus,abk"
Sprite 8,100,100,1
A*="L: IF X=300 then Jump L else X=X+1; Jump L"
Amal 8,A$
```

This program will generate a syntax error because IF will be interpreted as the two instructions I and F,, To find the position in the animation string of this error, type the following instruction from the direct window.,

```
Print f!id$(A$,Amalerr,Amaller + 5)
```

Error messages

If you make a mistake in one of your AMAL programs, AMOS will exit back to Basic with an appropriate error message,, Here's a full list of ths errors which can be generated by this system, along with an explanation of their most likely causes.

Bank not reserved;; This error is caused if you attempt to call the PLay instruction without first loading a bank containing the movement data into memory. This should be created with the AHAL accessory program. If you're not using

PLay at all then check that you've correctly separated *Any* Pause and Let instructions.

Insturction only valid in Autotest;; You've inadvertently called either the Direct or the eXit instructions from your main AI1AL program.

Illegal instruction in Autotests Autotest may only be used in conjunction with a limited range of AMAL commands. It's not possible to move or animate our objects in any way inside an autotest. So check for erroneous commands1ikeMove,,AnimorFor«„.Wext„

Jump To/Within Autotast in animation string: The commands inside an autotest function *&re* completely separate from your main AHAL program. So AHAL does not allow you to jump directly inside an ALitotest procedure. To leave an autotest,, and return to your main AMAL program you must use either eXit or Direct.

Label already defined in animation strings You've attempted to define the same label twice in your'A11ALprogram,,A11A11A!…1abe1sconsistofjustasingle CAPITAL letter. So "Test" and "Total" *&re* just different versions of the same label (T). This error is also generated if you have accidentally separated two instructions by a ":" (colon). Use a semi-colon instead,,

Label not defined in animation strings This error is generated when you. try to jump to a label which doesn't currently exist in your animation string.

Next without For in animation strings Like it's Basic equivalent each For command should be matched by a corresponding Next statement. Check any nested loops for an spurious Next command.

Syntax error in animation strings You've made a typing mistake in one of your animation strings. It's easy to cause this error by accidentally entering an AMAL instruction in full,, just like its Basic equivalent.

Animation channels

Amos allows you to execute up to 64 different AMAL. programs simultaneously. Each program is assigned to a specific animation channel.

Only the first 16 channels can be performed using interrupts. If you need to animate more objects you'll have to turn off the interrupts using SYNCHRO OFF. You can now execute the AMAL programs step by step using an explicit call to the SYNCHRO command in yur main program loop. As a default,, all interrupt channels are assigned to the relevant hardware sprite.

CHANNEL (assign an object to an AMAL channel)

C! -I A kIKl C" L. .-, T O C-J b _; «⇔ c +_

The CHANNEL command assigns an animation channel to a particular s<reen

related "object". In AMAL, you're not restricted to a single channel per object. Any single screen object can be safely animated with several channels if required. There are various different forms of this instruction.

Animating a computed sprite

CHANNEL n TO SPRITE s

This assigns sprite number s to channel n. As a default, channels 0-7 are automatically allocated to the equivalent hardware sprite, and 8-15 are reserved for the appropriate computed sprites.

In order to animate the computed sprites from 16 onwards, you'll need to allocate them directly to an animation channel with the CHANNEL command. As normal , sprite numbers from 8 to 63 specify a computed sprite rather than a single hardware sprite. For example:

```
        Channel 5 To Sprite 8 : Rem Animates Computed sprite 8 using
                                          Channel 5
```

The X,Y registers in your AMAL program now refer to the hardware coordinates of the selected sprite. Similarly the current sprite image is held in register A.

Animating a blitter object

CHANNEL n TO BOB b

Allocates blitter object b to animation channel n. This object will be treated in an identical way to the equivalent hardware sprite. The only difference is that registers X and Y now contain the position of your bob in screen coordinates.

Note that if you've activated screen switching with the DOUBLE BUFFER command, this will be automatically used for all bob animations.

Moving a screen

AMOS Basic allows you to freely position the current screen anywhere on your TV display. Normally this is controlled with the SCREEN DISPLAY instruction. However, sometimes it's useful to be able to move the screen using interrupts.

CHANNEL n TO SCREEN DISPLAY d

This sets the channel n to screen number d. Screen d can be defined anywhere in your program. You'll only get an error if the screen hasn't been opened when you start your animation.

The X and Y variables in AMAL now hold the position of your screen in hardware coordinates. Register A is *not* used by this option and you can't animate screens using Anim. Otherwise all standard AMAL instructions can be performed as normal. So you can easily use this system to "bounce" the picture aroud the display. Examples:

```
        Load  Iff  "AMOS  DATA:IFF/Frog_screen.  IFF",1
        Channel 0 To Screen Display 1
```

```
Amal 0,"Loops Hove 0,200,100 ; Hove 0,-200,100 ; Jump Loop"
•Amal On 0 s Direct

Load Iff "AMOS_DATA;IFF/Froq_screen,.IFF",1
Channel 0 To Screen Display 1
Rem Screen can only be displayed at certain positions in the X
Amal 0,"Loops Let X=XM; Let Y---YH; Pause; Jump Loop"
Amal On s Direct
```

For a further example of this technique, load EXAMPLE;: 14,,4,, This demonstrates how the SCREEN DISPLAY can be used in conjunction with the menu commands. to slide the menu screen up and down your display., It's similar to the display system found in Magnetic. Scrolls' excellent series of adventures.

Hardware scrolling

Although hardware scrolling can be performed using AMOS E<asic's SCREEN OFFSET command, it's often easiest to animate your screens using AMAL instead as this generates a much smoother effect.

CHANNEL n TO SCREEN OFFSET d

This assigns AMAL program number n to a screen d, for the purpose of hardware scrolling. The X and Y registers now refer to the section of the screen which is to be displayed through your TV. Changing these registers will scroll the visible screen *Area* around the display. Here's an examples

```
Screen Open 0,, 320,500,32, low res s Rem Open an extra tall screen
Screen Display 0,,45,320,250
Load Iff "AMOS_DATA:iFF/Magic..screen.IFF" ,
Screen copy 0,0,0,320,250 To 0,,0,251
Screen 0 s Flash Off s (Set Palette (0)
Channel 0 to Screen Offset. 0
Amal 0,"Loops Let X=XM-i28; Let Y=YM-45; Pause; Jump Loop"
Amal On s Wait Key
```

This program allows you to scroll through the screen using the mouse. Try moving the mouse in direct mode. For a further example of hardware scrolling, see EXAMPLE 14.5

Changing the screen size

CHANNEL n TO SCREEN SIZE s

This allows you to change the size of a screen using AMAL. s is the number of the screen to be manipulated. Registers X and Y now control the width and height of your screen respectively. They're similar to the W and H parameters used by the SCREEN DISPLAY command,, Examples

```
Load Iff "AflOSJ)ATA:IFFYMagic:screen.IFF" ,0
Channel 0 to Screen Size 0
Screen display 0,,,320,1 s Rem set the screen size to 1
Af-Loop; For R0=0 To 25S ; Let Y=RG ; Next ROs "
A$=A*+"For R0=0 To 254; Let Y=255-R0; Next RO5 J Loop"
Amal 0,A$ : Amal On s Direct
```

Rain bows

CHANNEL n TO RAINBOW r

This option generates a rainbow effect within an A HAL program. As usual n is the number of an animation channel from 0 to 63.. r is an identification number of your rainbow (0-3)«

X holds the current BASE of your rainbow.. This is the first colour of your rainbow palette to be displayed,, Changing it will make the rainbow appear to turn. Y contains the line on the screen at which the rainbow effect will start,, If you alter this value., the rainbow effect; will move up or down. All coordinates are measured in ^hardware* format,,

Register A stores the height of your rainbow on the screen. See the AMOS Basic RAINBOW command fore more details,,

Advanced tehcniques

The AUTOTEST system

Normally ail AMAL programs *are* performed in strict order from start to finish. Inevitably some commands such as Move and For,»..Next will take several seconds to complete. Although this will be fine in the vast majority of cases it may lead to significant delays in the running of certain programs. Take the following simple programs

```
        Load "AMOS.._DATA;;Sprites/Octopus,,abk" s Get Sprite Palette
        Sprite 8,130,50,1
        Amal 8,"Loop: Let RO^Xii-X:; Let Rl^YH-Y; Hove R0,Rl,50; Jump Loop"
        Amal On r. Direct.
```

As you move the mouse;, the sprite is supposed to follow it around on the screen. However in practice the response time is quite sluggish, because the new values of XII and Yi'i are only entered after the sprite movement has totally finished„ Try moving the mouse in a circle„ The octopus is completely fooled!

Autotest, solves this problem by performing your tests at the start of every VBL, before continuing with the current program. You tests now occur at regular 1/50 intervals., leading to a practically installtanous response:

Auto test commands

The syntax of Autotest is;

AUtotest (tests)

"tests" can consist of any of the following AMAL commands.

Let reg=exp

This is the standard AMAL. Let instruction, It assigns the result of an expression to register "reg",.

Jump label

The Jump command jumps to another part of the current autotest. "label" is defined using the colon "s" and KHUST* lie inside the autotest brackets,,

eXit

Leaves the autotest and re-enters the main program from the point it left off,,

Wait

Wait turns off the main AI1AL program completely, and only executes the Autotest.

If

In order to simplify the testing process inside an autotest routine there's a specially extended version of the AHAL If statement* This allows you to perform one of three actions depending on the result of the logical expression "exp".

```
If exp Jump L    (Jumps to another part of the autotest)
If exp Direct L  (Chooses part of the prog to be executed after AU)     201
If exp eXit      (Leaves autotest)
```

On

Restarts the main program after a previous Wait instruction,, This lets you wait for a specific event such as a mouse click without wasting processor time.

Direct label

Direct changes the point at which the main program will be resumed after your test. AMAL wi11 now jump to this point automatica11y at the next vertical blank period. Note that label fctnust* be defined outside the Autotest brackets.

Inside Autotest

Here's the previous example rewritten using the Autotest feature

```
Load "AiiOS._DATAsSprites/octopus,,abk"
Sprite 8,130,, 50,1 s Get Sprite Palette
A$="AUtotest (If ROOXil Jump Update"
A*-A$+"lf R1OYPI Jump Update else eXit"
A$^A$+"Update:; Let R0=XM; Let Ri.~Yi1; Direct 11)" s Rem End of AD
A*=A$ + "!1s Move RQ~X.R1--Y,,2O Wait;" :: Rem Try changing 20 to
                                               different values!
Arnal 8,A$ s Amal On
```

The sprite now smoothly -follows your mouse, no matter how fast you move it. The action of this program is as follows:

Every 50th of a sec the mouse coordinates &TB tested using the XM and YM functions. If they are unchanged since the last test, the Autotest is aborted using the eXit command. The main program now resumes precisely where it left off.

However if the mouse has been moved, the autotest routine will restart the main program again from the beginning (label It) using the new coordinates in XII and YM respectively,

UPDATE EVERY (save some time for your Basic prog rams)

UPDATE EMERY n

Although most ARAL programs *are* performed practically instantaneously, any objects they manipulate need to be explicit;/ drawn on the Amiga's screen.

The amount of time required for this updating procedure is unpredictable and can vary during the course or your program., This can lead to an annoying jitter in the movement patterns of certain objects™

The UPDATE EVERY command slows down the updating process so that even the largest object can be redrawn during a single screen update., This regulates the animation system and generates delightfully smooth movement effects,,

n is the number of vertical blank periods between each screen update. In practice you should start off with a value of two, and gradually increase it until movement is smooth.

One useful side effect of UPDATE EVERY, is to reserve more time for Basic to execute your programs™ With a judicious use of this instruction, it's sometimes possible to speed up your programs *by* as much as 302, without destroying the smoothness of your animation sequences,,

SYNCHRO (execute an ANAL program directly)

SYNCHRO [ON/OFF]

Normally AMOS Basic will allow you to execute up to 16 different AMAL programs at a time., This limit is determined by the overall speed of the Amiga's hardware. Each AMAL program takes its own slice of the available processor time. So if you're using the standard interrupt system, there's only enough time to execute around 16 separate prog rains„

The SYNCHRO command allows you to exceed this restriction by executing your AMAL programs directly from Basic. Instead of using interrupts, all AMAL programs are now run using a single call to the SYNCHRO command., Since AMAL programs execute far faster than the equivalent Basic routines, your animations will still be delightfully smooth,. But you will now able to d«•cide when and where yur A11AL routines will be performed in your program.

One additional bonus is that you can now include collision detection commands such as Bob Col or Sprite Col directly in your AMAL routines. These are not available from the interrupt system as they make use of the Amiga's blitter chip. This would be impossible using iterrupts.

£? & f o r G? c a l l in a SVKlCi-iPO yir,t.x -f'i rs-^ n<»e* d '{-. <-i h i r f i c%-p-f t h o :!. ,-, +. <• i- y i.i p -L *r, w :i, \ h
SYNCHRO OFF. It's imporatnt to do this *before* defining your AMAL programs, otherwise you won't be allowed to use channel numbers greater

than 15 without an error.

Due of the sheer power of the animation system, it's nearly possible to write entire arcade games completely in AMAL. This leaves your Basic program with simple jobs such as managing the hi-score table and loading your attack waves from the disc. The results will be indistinguishable from pure machine code, A good example is Cartoon Capers, the first commercial games release that's written entirely in AMOS.,

A demonstration of SYNCHRO can be found in EXAMPLE 14,6.

STOS compatible animation commands

The original STOS Basic included a powerful animation system which allowed you to move your sprites in quite complex patterns using interrupts. At the time, these commands were hailed as a breakthrough™

Although they've now been overshadowed by the AMAL system, they do provide a simple introduction to animation on the Amiga. So AMOS provides you with the entire STOS animation system as an extra bonus!

If you're indenting to convert STOS programs to AMOS., you'll need to note the following points:

* Unlike STOS, the movement patterns in AMOS Basic can be assigned to any animation channel you like. The Hove commands can therefore be used to move bobs, sprites or screens, using exactly the same techniques.

  As a default, all animation channels *are* assigned to the equivalent hardware sprites. In practice you may find it easier to substitute blitter objects as these are much close to the standard STOS Basic sprites. Add a sequence of CHANNEL commands to start of your program like so:

  ```
  Channel 1 to bob 1
  Channel 2 to bob 2
    :       :
  ```

  Don't forget to call DOUBLE BUFFER during your initialisation procedure, otherwise your bobs will flicker annoyingly when they're moved.,

* The same channel can be used for both STOS animations and AMAL programs., So it's easy to extend your programs onee they've been succesfully converted into AMOS Basic. The order of execution is;

  ```
  AMAL
  MOVE X
  MOVE Y
  ANIM
  ```

MOVE X (move a sprite horizontally)

MOVE X n,m$

Defines a list of horizontal movements which will be subsequently performed on animation channel number n.

n can range from 0 to 15 and refers to an object you have previously assigned using the CHANNEL command. m$ contains a sequence of instructions which together determine both the speed and direction of your object, These commands are enclosed between brackets and are entered using the following format;

```
(speed,step,count)
```

There's no limit to the number of commands you can include in a single movement string, other than the amount of available memory.

"speed" sets a delay in 50ths of a second between each successive movement step. The speed can vary from 1 (very fast) to 32767 (incredibly slow).

"step" specifies the number of pixels the object will be moved during each operation,, If the step is positive the sprite will move to the right,, and if it is negative it will move left.

The apparent speed of the object depends on a combination of the speed and step size. Large displacements coupled with a moderate speed will move the object quickly but jerkily across the screen.. Similarly a small step size combined with a high speed will also move the object rapidly, but the motion will be much smoother,, The fastest speeds can be obtained with a displacements of about 10 (or -10).

"count" determines the number of times the movement will be repeated,. Possible values range from 0 to 32767. A count of 0 performs the fnovernent pattern indefinitely.

In addition to the above commands, you can also add one of the following directives at the end of your movement string.

The most important of these extensios is the L instruction (for loop), which jumps back to the start of the string and returns the entire sequence again from the beginning. Example:

```
Load "A!10S._.DATAsSprits/0ctopus,,abk" : Get Sprite Palette
Sprite 1,130,100,1 : Rem Define Sprite 5
Move X l,"(l,5,A0)(l,-5,,60)L"
Move On
```

The E option allows you to stop your object when it reaches a specific point on the screen., Change the second to last line in the above example to;

```
Move X l,"(l,5,30)E100"
```

Note that these end-points will only work if the x coordinate of the object exactly reaches the value you originally designated in the instruction. If this increment is badly chosen the object will leap past the end-point in a single bound,, and the test will fail,, Example:

```
Load "Ai10S_DATA:Sprites/0ctopus.abk" s Get Sprite Palette
Channel 1 To Sprite 8 : Channel 2 To Sprite 10
Print At(0,5)+"Loop:irtg OK"
Sprite 8,130,100,1
Move X 1," (1,10,30) (1,,-10,30)1..."
Move On
Print At(0,10)+"l\low press a key" : Wait Key
Sprite 10,140,150,2
Move X 2,"(li.i5,26)L" s Move On 2
Print At < 0,, 15) + "Oh dear!" ;; Wait Key
```

MOVE Y (Move an vertical object)

HOVE Y n,m$

This instruction complements the MOVE X command by enabling you to move an object vertically along the screen. As before, n refers to the number of an animation sequence you've allocated using the CHANNEL commandj, and ranges between 0 and 15.

m$ holds a movement string in *An* identical format to HOVE X, Positive displacements now correspond to a downward motion, and negative values result in an upward movement. Examples;

```
        Load "AM0S_DATA:Sprites/0ctopu5.abk" : Get Sprite Palette        205
        Channel 1 to Sprite 8 : Sprite 8,130,,10,1
        Move Y 1,"10(1,1,180)1."
        Channel 2 To Screen Display 0
        Move Y 2,"(l,4,25)(l,-4,25)
        Move On s Wait Key
```

MOVE ON/OFF (start/stop movements)

MOVE ON/OFF [n]

Before your movement patterns will be executed they need to be activated using the MOVE ON command.

"n" refers to the animation sequence you wish to start, and can range f rom 0 to 15. If i t' s omi tted then a.11 your movernen ts will be activated simultaneously.

HOVE OFF has exactly the opposite effects It stops the relecant movement sequences in their tracks.

HOVE FREEZE (temporatily suspend sprite movements)

HOVE FREEZE [n]

The MOVE FREEZE command temporarily halts the movements of one or more objects on the screen. These objects can be restarted again using HOVE ON.

"n" is completely optional and specifiew the number of a single object to be suspended by this instruction,

=MOVON (return movement status)

x=MOVON(n)

MOVON checks whether a particular object is being moved by the MOVE X and MOVE Y instructions. It returns -1 if object n is in motion, and 0 if it's stationary. Do not confuse this with the MOVE ON command. Also note that MOVON searches for movement patterns generated using the MOVE coii«and5, so it will **not** detect *Any* animations qenerated by nIAL «

A NIP! (animate an object)

AMI PI n,af

Automatically flicks an object through a sequence of images creating a smooth animation effect on the screen. These animations are performed 50 times a second using interrupts, so they can be executed simultaneously with your Basic programs,

"n" is the number of the channel which specifies a sprite or bob to be animated by this instruction.

"a$" contains a series of instructions which define your animation sequence. Each operation is split into two separate components enclosed between round brackets,

"image" is number of the image to be displayed during each frame of the animation, "delay" specifies the length of time this image will be hied on the screen (in SOths of a see.),, Examples 206

```
        Load "AM0S_DATA:Sprites/(3ctopus.abk" s Get Sprite Palette
        Channel 1 to Sprite 8 s Sprite 8?200,,100,,1
        Anim 1," (1, .1.0) (2,10} (3,10) (4,10)"
        Anim On ; Wait Key
```

Just as with the MOVE instruction, there's also an L directive which enables you to repeat your animations continuously. So just change the ANIM command in the previous example to the following!

```
        Anirn i,"(l,10)(2,10)(3,10)(4,10)L"
```

ANIM ON/OFF' (start an animation)

ANIN OM/OFF [n]

AKIIPI OKI activates a series of animations which have been previously created using the AW III command,, n specifies the number of an individual animation sequence to be initialised,, If it's omitted, then all current animation sequences will be started immediately,

A.MIPi OFF [n]

Halts one or more animation sequences started by ANIM ON.

ANIM FREEZE (freeze an animation)

ANIM FREEZE [n]

Temporarily freezes the current animation sequence on the screen,, n chooses a single animation sequence to be suspended. If it's not included, all current animations will be affected. They can be restarted at any time with a simple call to the ANIM ON instruction, Nowadays, it's not uncommon for an *arc&ds* gaffie to contain himderds of different screens. With compaction,, it's possible to crap a single 32 colour screen into about 30k of memory. So 100 screens would be the equivalent of about 3 Megabytes of data. Imagine how difficult this would be to fit into a standard A50Q!

The classic way of avoiding this restriction, is to construct your backgrounds out of a set of simple building blocks. Once these "tiles" have been created, they can be placed on the screen in any order you like. So the same set of tiles can be reused to generate a vast number of potential screens. Each screen is now stored as a simple list of its components, and requires a tiny fraction of the original memory,,

In order to exploit this system, you'll obviously need some way of defining your various screen maps. As you might have guessed,, we've helpfully provided you with a powerful map definer accessory on the AMOS program disc. Full details can be found in the accompanying documentation file.

AMOS Basic also includes a number'of special instructions for drawing your tiles on the screen,, These make it easy to generate the fast scrolling backgrounds that *&re* the hallmark of a modern arcade game™

Icons

Icons are separate images which have been especially designed for producing your background screens. Once you've drawn an icon, it's fixed permanently into place. So you can't move it to a new position using the AMAL animation system.

All icons *ars* stored in their own AMOS memory bank (M2). This bank is created using the Sprite definer accessory (on the AMOS Program disk), and will be automatically saved along with your Basic programs.

Like Bobs, Icons are displayed using the Amiga's amazing Slitter chip. But since Icons are essentally static objects, they are usually drawn in REPLACE mode. Your icons will therefore totally erase any existing graphics at the current screen position.

PASTE ICON (draw an icon)

PASTE ICON x,y,n

Draws icon number n on the screen at GRAPHIC coordinates x,y,. n is the number of the icon which is to be displayed. This must have been previously stored in the ICON bank.

Icons can be freely positioned anywhere on the screen,, subject to the normal clipping rules. Examples

```
Load "Ai10S..,.DATA5lcons/Nap__icons.abk"
Screem Open 0,320,256,32,Lowres s Cls 0 s Get Icon Palette
For X=l To 11 s Paste Icon X*32,0,l : Next X
FOK- V-l To A a P.r,mt<» I <; o i. O,Y*32+li a P«:s- I. e Icon £Be,Y*SS,l
Next Y
For X=l To 11 : Paste Icon X*32,223,1 : Next X
```

Note that if you're using double buffering., a copy of your icons will be drawn into both the physical and logical screens. Since this is rather slow, it's common practive to add a call to AUTQBACK 0 before drawing your icons on the screen,, This restricts straight to the physical screen using SCREEN COPY,, saving a considerable amount of time.

For a further example, see the MAPVIEW program on the A11 OS DATA diss.. This displays a background screen you've created using the AMOS Map Editor.

GET ICON (create an icon) 2,08

GET ICON [s,,] i,tx,ty TO bx,by

Captures an image from the screen and loads it into icon "i". If this icon does not presently exist, it will be created for you in bank 2,, This bank will be automatically reserved by the system if required.

i is the number of your icon, starting from 1. tx.ty to bx,by define the rectangular zone which encloses the selected region.

s determines the number of the screen which will be used as the source of your image. If it's omitted, the image will be taken from the current screen instead,, Example;

```
Erase 2
F*=-Fsel*("#. *".,"".,"Load a screen") : If F*="" Then Direct
If Exist(f$) Then Load Iff f*,0 Else Direct
SH=Screen Height : H=SH/32-l : SW=Screen Width : W==SUi/32~i
For Y=0 to H
  For X=0 to W
    Get Icon X+Y*W+1,X*32,Y*32 To X*32+3i ,Y*32+3i
  Next X
Next Y
Cls 0
Do
  Paste Icon Rnd (Sw-1) ,Rnd(SH-»I) ,Rnd/(H*W)+l
Loop
```

GET ICON PALETTE (get icon colours)

GET ICON PALETTE

Grabs the colours of the icon images in bank 2, and loads them into the current screen palette,, This command is normally used to initialize the sc:reen after you'be loaded some icons from the disc„ Exarnp1e:

```
Load "Ai10S..,.DATR:Icons/l!ap..,icons,,abk"
Get Icon Palette
Paste Icon 100,100,1
```

DEL ICON (deletes icons) 209

DEL ICON n[TO m]

Deletes one or more icons from the icon bank, n is the number of the first icon to be removed.

(n is the optional number of the last icon to be deleted in the list, if it's included all the icons from first to last will be erased one after another.

When the final icon in a bank has been deleted, the entire bank will be removed from memory,,

MAKE ICOW MASK (set colour zero to transparent)

MAKE ICON MASK [n]

Normally, any icons you draw on the screen will completely replace the existing background. The icon will seem to be displayed in a rectangular box filled with colour zero.

If you want to avoid this effect and overlay your icons directly over the current graphics, you'll need to create a *mask* for your icons. This informs AMOS that colour zero should be treated as transparent.

n is the number of the icon to be affected. If it's omitted,, a mask will be defined for all icons in the bank. See EXAMPLE .1.5.1

Screen blocks
=============
AMOS Basic supplies you with a set of powerful BLOCK commands which allow you to grab part of an image into memory and paste it anywhere on the screen.

These instructions *are* mainly used for holding temporary data, since your blocks cannot be saved along with your Basic programs.

Blocks *are* especially effective in the construction of dialogue boxes, as they can be used to save the background areas before displaying your new graphics.

They can also be exploited in puzzle games like Split Personalities. Each block can be loaded with a single section of your image,, You can then jumble your pictures by rearranging the blocks on the screen with PUT BLOCK.

GET BLOCK (grab a screen block into memory)

GET BLOCK n,tx,ty,,w,h[smask]

GET BLOCK grabs a rectangular area in block number n, starting at coordinates tx,ty.

n is the number of the block ranging from 1-65535,, tx,, ty set the coordinates of the top left hand corner of your block. w,y hold the width and height of the block respectively,,

"mask" is a flag which chooses whether a mask will be created for yourr,e?wb1oc:k„

mask~-0 Replace mode. When the block is drawn on the screen,, it will totally destroy any graphics at that current position,

fliask~i Calculates a mask for the block. Colour zero will now be treated as if it were transparent,,

PUT BLOCK (copies a previously created block onto the screen) 210

```
PUT BLOCK n[,x,y]
PUT BLOCK n,x,y,plaries[,minterms]
```

PUT BLOCK copies block number n to the current screen,, x,,y specify the position of your new block on the screen. If they are omitted the block will be redrawn at its original screen coordinates,,

Note that all drawing operations will be clipped to fit into the current screen,, starting from the nearest 16 pixel boundary.

For a demostration of the BLOCK commands see the routine in EXAMPLE 15.2. We've also provided experienced programmers with a couple of optional extras. These a.re not needed for the vast majority of applications, they're only required when you want to achieve weird special effects on the screen!

"planes" holds a bit-map which sets the range of colours which will be drawn in your block,, The Amiga's screen is divided up into segments known as bit-planes. Each plane contains a single bit for every point on the Amiga's screen. When the Amiga's hardware displays this point, it combines the bits from each plane to calculate the required colour number. Each bit in "planes" represents the status of a single bit-plane. If it's set to one, then the selected plane will be drawn by the instruction,, otherwise it will be completely ignored. The first plane is represented by bit zero,, the second by bit one,, etc,

Usually, the block will be displayed in all the available bit-planes., The corresponds to a bit-pattern of %111111

"fitinterm" selects the blitter mode used to copy your block on the screen. A full description of the possible drawing modes can be found in the section on SCREEN COPY, The best way to loearn about these options is to experiment!

DEL BLOCK (delete a screen block)

```
DEL. BLOCK n
```

Deletes one or more blocks and restores the memory used to AMOS Basic.

DEL BLOCK	Erases *all* current blocks
DEL BLOCK n	Deletes block number n.

GET CBLOCK (save and compact a screen image) 21.1

```
GET BLOCK n,x,y,sx,sy
```

The GET BLOCK command saves and compacts a rectangular area of the screen. The compaction system used by this, command has been especially

If you've used the Amiga for some time you'll already be familiar with the idea of menus. Impossible as it seems, AMOS has taken the existing system and improved it almost beyond recognition.

Menus can be created with up to eight separate levels, and each individual menu item can be repositioned on the screen at will. Menu titles can be printed in any combination of colours or styles. You can also include bobs or icons directly in your menus using an amazing menu definition language,,

AMOS Basic is squally impressive when it comes to reading, a menu,, There's a buit-in interrupt-dricen ON MENU command which can automatically branch to a selected point in your program depending on the option selected,, Furthermore, any menu option can be accessed directly from the keyboard using the MENU KEY instruction.

For a demonstration of the terrific effects that can be achieved with this system;, load the program EXAMPLE 16.1.

Using a menu

All AMOS menus &re called up by holding down the right mouse button in the standard way,, Ones a menu has been activated you can then select an option directly with the mouse cursor. When you release the button, the option number you have chosen will be returned to your program,

Menus can be repositioned by placing the mouse cursor over the top left corner of an item and holding down the LEFT button, A small box will now appear on the menu bar which can be dragged across the screen using the mouse,

In addition, holding down the SHIFT key will freeze a menu into place. This allows you explore a menu without selecting any of the various options. You can also use any of the mouse features such as slowing or axis selection in conjunction with your menus.

Creating a simple menu

AMOS menus can be created either directly within your programs or using a special menu definer included on the AMOS program disc.

If you've never used menus before, the? sheer variety of the available fflenucoA)mandsmayseema1i111eoverwhe1ming,.Here'sabrief description of the basic features to provide you with a painless introduction to AMOS menus.

Setting the title line

The first stage in the creation of a menu is to define the "ti11e line". THe title line of a menu can be set using the I1£NU$ command. In its simplest form this has the formats

```
MENU* (set a menu title)
```

MENU$(n)=title*

MENU* creates a title line for your menu. Each heading is assigned it's own individual number starting from one, and increasing from left to right. So the leftmost title is repsresented by a one, the next title as two, etc.

The text in "title*" holds the name of the option which will be displayed in your new menu,. Here is a simple example which constructs a menu line consisting of just two titles; ACTION and MOUSE

```
        Menu*(l)=" Action "
        Henu*(2)   =   "   Mouse   "
```

Note the space after "Action" - this will separate it from Mouse, the next menu along. You must now specify a list of options to be associated with each of your new headings. These form a vertical bar which will drop into place whenever a title is selected with the mouse,,

MENU$(t,o) (set a menu option)

MENU$(t.,o)=option*

This second form of MENU* defines a set of options which will be displayed in the menu bar,.

t is the number of menu heading which your option will displayed under, o is the option number you with to install in the menu bar. All options are numbered downwards from the top of the menu,, starting from one.

The only physical limit to the size of your menu is the amount of memory, but it's wise to restrict, yourself to less than about 10 options for each title. This will keep the complexity of your menus down to an agreeable minimum.

"option*" holds the name of your new option,, This can consist of any section of text you like.. For an example, try adding the following lines to the program above;

```
        Rem Action menu
        Menu*(1,1)=" Quit "
       •Rem House menu
        Menu*(2.,1) = " Arrow "
        Menu*(2,2)=" Pointer "
        Menu*(2,3)=" Clock "
        Wait Key
```

This specifies a list of alternatives for the ACTION and the mouse menus. If you try to run this program as it stands, nothing will happen. That's because the menus need to be initialised with a call to the MENU ON command. Enter this thin above program before the Wait Key instruction. Now run the example and select the menu items with the mouse cursor., Remember to hold down the RIGHT mouse button first!

MENU ON (activate menu)

MENU ON

Activates a menu defined using the MENU* command. The menu line will now appear automatically when ths right mouse button is pressed ny the user. To start the previous menu, insert the following line after the definition statements.

```
        Menu On
```

Go to the Direct window and play around with the menus. Select options by pressing the right mouse button 214

Reading a simple menu
=====================

Once you've created your menu and activated the AMOS menuing system you'll want to discover which options have been selected by the user. This can be accomplished using a simple form of the CHOICE command.

=CHOICE (read a menu)

selected=CHOICE

CHOICE returns a value of -1 (true) if the menu has been highlighted by the user, otherwise 0. It's automatically reset to 0 after each test. It's also possible to find the title number which has been selected using a second form of this instruction.

heagind=CHOICE(l)

"heading" now contains the number of the "title" which has been highlighted by the user. Similarly you can retrieve the actual option number which has been chosen with a parameter of two.

item=CH0ICE(2)

Try adding the following lines to the previous examples

```
        Do
          Rem If choice=-! can be simplified to: If choice, as seen,,,.
          If choice and choice(i)=l Then Exit
          If choice(l)=2 and choice(2)<>0 Then Change Mouse choice(2)
        Loop
```

This changes the shape of the mouse cursor depending on which option you have chosen from the menu. A full demonstration of these menu can be found in the file EXAMPLE 16.2.

Advanced menuing features

We will now cover some of the more advanced menuing features available from within AMOS Basic. Used properly these A110S menus can add a whole new dimension to your programs.

MENU* (create a menu) 215

M!~ ML! * <;,;,) = nan « a l 4 [..sj◆1ec:-t&d*3l" ,,inac:t:i,ve*"} \~ ,t> acks:|round*!3

Defifies the appearance of each individual menu item in one of your menus« Unlike normal Amiga menus these items are not restricted to standard text. They can also include embedded commands which allow you to draw bobs., icons or graphics at any point in the menu line,.

Any of the parameters in this instruction *m&y* be optionally omitted, so you can change parts of a menu description independently. A value of "" in your menu string will ERASE the existing setting. Similarly you can retain the original value by including a comma at the appropriate point., For example:

```
        Menu$(l)-" Action ",,"" s Rem Erase second option
        Menu$(2)=" Mouse 2 ",, s Rem Change title without altering
                                     anything else.
```

The position of the menu item within the actual menu is indicated using a list of up to eight parameters separated by commas., The general format iss

```
        {item)/(item„opti0n)/(item,option,,suboption)...
```

"normal*" is a string which sets the normal appearance of an item when it's displayed in the menu, "selected*" changes the effect of highlighting a menu option with the mouse,, As a default,, selected items areprintedininversetext„

"inactive*" changes the appearance of an item which has been deactivated using the MENU INACTIVE command. If this string is omitted., all inactive imtes will be displayed in italics, "backgrounds" creates a background for your menu items when they *&re* initially drawn. Generally this will be a bo of some sort created with the internal Bar or line commads.

For now one, we'll abbreviate these parameters using a standard notation:

```
        setting$=[jselected*]L,inactive*][,background*]
```

1fhe menu hierarchy

The level of an item in the menu is determined by its position in the menu hierarchy.

```
        Menu*(i)="Title"
        Menu*(I,i)="0ption 1"
        Menu*(l,2)="0ptian 2"
        Menu*(i,2,1)="Item 1"
```

This defines a simple menu. The structure of a menu is similar to that of an *a.rr&y*. Each level of the menu is represented by its own dimension in the array, and is controlled using a separate version of the MENU* command»

The first level represents the title line which appears at the top of your menus. It can be set using a command likes

```
        Henu$(n)=title*[setting$Ii
```

"n" now corresponds to the position of the title'from the left of the screen, and setting* refers to the three optional strings which define the general appOAF-Ar^t-K [illegible] menti. [illegible]
of your menus first as this ^dimensions* the *&rr&y,* All other items may be created in any order you. wish.

Each title is associated with a list of menu options which drop into view when the menu is selected,, These form the second level of the menu structure and &re defined using a second version of the MENU* command,,

```
        Henu$(n,option)=Item*[setting$.]
```

"option" holds the number of the item measured from the top left of the menu bar. There's no limit to the number of options which may he linked to a single title, other than the amount of available memory.

Each individual option can in turn be associated with its own sub menus up to a total of eight levels,,

```
        IIenu$(n, option,,sub option )=Item$[setting$]
```

Once you've created a menu it can be expanded or charmed at any point in your program,, Never change the current screen while you a.re creating a menu as this will lead to an error message.

See EXAMPLE 16.3

=CHOICE (read menu)

item=CHO.ICEC(dimension)]

The CHOICE function checks whether an option has been highlighted on the current menu. If an item has been selected (down to the lowest 1evel),CH01CEwi11returnava1ueof-1,otherwiseitwi11be0„After you've called this f unc ti on, the s ta t us o f t he m enu w i11 be automatically restored to 0 (false). This stops a single menu, access from being accidenta11y detected severa1 times„

The second form of this command returns the option selected at the required level.

itefn-"CMOICE(dimension)

"dimension" indicates the level of the menu which is to be read. As you m&y recall, a level number of 1 corresponds to the title line of the menu. Similartly the levels between 2 and 8 indicate the number of an icption which has been chosen,, If a menu item has not been selected, "item" will be loaded with a value of zero,, For example:

```
        Menu*<i)="Menu"
        Menu*(1,1)="Option 1"
        Menu*(l,2)="0ption 2"
        Menu*(l,2,l)"~"0ption 2.1"
        Menu On
        Do
          If choice Then Print choices 1),, choic:e(2),, choice(3)
        Loop
```

If you wanted to implement larger menus with this system,, your program would need to use a long list of IF,,.,.THEM statements to deal with each and every possibility,, This would cause a small but significant delay in your program while the menus were being read, It would also make it very difficult to amend your program later,, Fortunately AMOS Basic

OKI MENU PROC proci [,proc2,...]

Each title in your menu can be assigned its own procedure which will be executed automatically whenever an option is selected by the *user,*, The action of this command is similar to the code below;

```
If Choice
  If Choice(i)=l
    Proci
  Endif
  IfChoice(i)=2
    Proc2
  Endif
  :
  :
Endif
```

There is one crucial difference between the OK! MENU command and the above instructions. ON MENU is performed 50 times a second using interrupts and does not affect the overall running of your program. This means that your program can be doing something totally different' while the menus are being checked by the system,,

Whenever the user selects a menu item the required procedure will be immediately executed with no further ation on the part of your program. Your procedure can then use the CHOICE command to find which option has been chosen and perform the appropriate action.

After the procedure has concluded, your program will be returned to the instruction following the ON MENU call. Here's an examples

```
Menu$(i)="Action" : Menu$(l,1)--"Count" s Menu$(1,2)="Quit"
Menu On : Rem Activate menu
On Menu Proc ACTION
On Menu On : Rem Activate On Menu command
Do
  X*=Inkey$ ; If X$<>"" Then Print X*;: Inc kl
Loop
Procedure ACTION
  Shared W
  If Choice(2)=l
    Locate 0,0 ; Print "You typed "jW;" letters" ; W=0
    On Menu On : Rem Initialise menus
  Endif
  If Choice(2)=2 Then Edit
End Proc
```

There are a couple of important points to note about this example. Firstly, see how the on menu sequence is activated using the ON MENU ON command. This *must* be called after the menu handling procedure has finished as it's needed to restart the menuing system. Also note the use of INKEY$ rather than INPUT. The INPUT command will halt the menu checks while you *&re* entering a line. All other commands can be used without problems, including WAIT, WAIT VBL and WAIT KEY. For a further example see EXAMPLE 16.4

ON MENU GOSUB (automatic menu selection)

ON MENU GOSUB label! C,label2,.....]

Enters one of a list of subroutines depending on the option which has been selected by the user,, Once you've called this command and created your subroutines, the menus will be checked automatically 50 times a second» •

Note that each title on the menu line is handled by its own individual subroutine. This differs from its AMIGA Basic equivalent which controls the entire menu with just a single routine.

After using this command you should activate the menuing system with a call to the ON MENU. The menus must be reinitialised in this *w&y* before jumping back to the main program with RETURN. Also note that label #11 AY NOT* be replaced by an expression as the label will only be evaluated once when the program is run.

ON MENU GOTO (automatic menu selection)

ON MENU GOTO label! [,label2,...]

This command has now been superceded by the more powerful ON MENU PROC and ON MENU 60SUB instructions. It's intended to provide com pa bil.it y with programs written in STOS Basic, itiHen ever a menu is selected., the program will jump to the appropriate label,,

ON MENU ON/OFF ([deactivate automatic menu selection)

ON MENU ON

Activates the automatic menuing system created by the ON MENU PROC/GOSUB/GOTO commands. After a sub-routins has been accessed in this way, the system will be DISABLED. So it's vital to reactivate the system with ON MENU ON before returning to the main program.

ON MENU OFF

This temporarily freezes the automatic menuing system,, It's useful when your program is executing a procedure which needs to be performed without interruptions - such as loading and saving information to the. disc. The menus can be reactivated using ON MENU ON,,

ON MENU DEL (dlete the labels used by ON MENU) *

OKI MENU DEL

This erases the internal list of labels or procedures created by the ON MENU commands™ You can now redirect your menus to another part of your program using a further call to ON MENU. WARNING! Only use this command after you've deactivated the menus with ON MENU OFF.

Keyboard shortcurs

Despite the undoubted appeal of menus, some users prefer to call up the Options of y< pragrAffl strAigKl -from -l.*<2> U:<^ybo*iti-<i.. AlihQwght me?n\-ts -.*rv? certainly easy for beginners, once you've familiarised yourself with a program it can be much faster to call up an option from the keyboard.

AMOS Basic: allows you to assign a keyboard shortcut to any of your menu items. These keystrokes *are* interpreted exactly as if the user had accessed the equivalent option from the menu. They can be used with any of the AMOS Basic menuing commands, including ON MENU.

MENU KEY (assign a key to a menu item)

```
MENU KEY(,,) TO c$
MENU KEYC,,) TO scan[,shift]
```

This allows you to assign any key to any item in a previously defined menu. The only restriction is that item you have specified must be at the bottom level of our menu. So you can't use a shortcut to select a sub menu as each command must correspond to a single option in the menu,.

c$ is a string containing a single character which is to be assigned to the menu option. Any additional characters in the string will be ignored.

Each key on the Amiga's keyboard is assigned its own individual scancode. By using this code you can assign keys to a menu which have no Ascii equivalents. Here is a list of scarscodes which can be used with your menus.

Scancode	Keys
80 ~ 89	Function keys F1-F10
95	Help
69	Esc

"shift" is an optional bitmap which allows you to check for control key combinations such as ALT+HELP or CONTROL (-Q. The format of "shift" is; 220

Bit	Key Tested	Notes
0	Left SHIFT	Only one shift key can be tested at a time
1	Right SHIFT	
2	Caps Lock	Either ON or OFF
3	CTRL	
4	Left ALT	
5	Right ALT	
6	Left AMIGA	C= key on some keyboards
7	Right AMIGA	

Note that if you set more than a single bit in this pattern., you'll have to press several keys simultaneously to call up your menu item,, Any of these short--cuts can be deactivated by using MEWL! KEY with no parameters. For examples

```
Menu Key(1,,10)
```

With the help of MENU KEY command,, adding shortcuts to a menu is a trivial operation, so you are strongly recommended to include them as standard in your programs. Here is an example that checks for the Amiga's 10 function keys;

```
Menu* i 15 =•" Fiinction k..^^ "
For A=l To 10
  OPT$=" F"+Str$(A)+" "
```

```
    flenu$(l,A)="OPT$
    Menu Key(i,A) To 79+A
  Next A
  Menu On
  Do
    If Choice Then Print "You pressed function key ";Choice(2)
  Loop
```

Menu con t roI c o m*m*a n *6*s

MENU ON (activate a menu)

MENU ON [bank]

Activates a menu which has been previously defined in your program. The menu **will** be displayed when the user next presses the right mouse button, and the options can be selected in the usual way. If a "bank" number is included with the instruction, then the menu will be taken from the appropriate memory bank,, See HAKE!! MENU BANK for more details.

MENU OFF (temporarily deactivate a menu.)

MENU OFF

THis is the opposite of the MENU ON command. It temporarily freezes the action of the entire menu. The menu can be restared at any time using the MENU ON command.

MENU DEL (delete one or more menu items)

Erases the selected menu from the Amiga's memory and restores the space to the rest of your program. There are two possible formats.

fiENU DEL

Erases the enitre menu. WARNING! This command is irrevocable!

MENU DEL(,,j,}

Deletes just a section of the menu. The (,,,) parameters contain a list up to eight values separated by commas. These indicate the precise position of the item in the menu hierarchy. For example;

```
  Menu Del(.t)  : Rem Erase title number 1
  Menu Del(1,2)  : Rem Erase option 2 of title i
```

MENU TO BANK (save the menu definitions
in a memory bank)

MENU TO BANK n

Thi.G ir.etniriion Alloue you *e> m.,v>-^«» .m-. entire m&•o11 -t!"•.-&• into memory bank n. If bank n already exist, you'll get a "bank already reserved" error.

  Once you've stored a menu in this way,, it will be saved automatically along with your Basic program. By storing your menu definitions in a memory bank, you can reduce the size of your program listings significantly. This will free valuable space in the editors memory, and will allow you to write longer Basic programs using exactly the same amount of memory.

BANK TO MENU (restores a menu definition
saved in a menu bank)

BANK TO MENU n

Sets up a menu definition from menu data stored in bank number n. You menu will be restored to exactly the same state as it was originally saved. If the menu is complex, this process may take a little time- To activate your nemenu ca11 the IIEMU 0M instruction.,

MENU CALX (recalculate a menu)

MENU CALC

One of the nicest features of AMOS menus is that they can be easily changed during the course of a progracn. After you've created your initial definition you can add new items and replace existing options as well,,

  Al your menu items &rs automatically repositioned when the menu is selected with the right mouse button,, If your menus are extremely large this may takek a little time. MENU CALC allows you to perform this process at the most appropriate point in your program, And avoid unnecessary and unwanted delays.

  Note that in order to stop the user calling the menu while it's being changed., you Are strongly adviced to freeze the menus with MENU OFF at the start of your procedure. The menu can then be safely restarted using the MENU ON command after you've finished. Evolving menus Are particularly useful for adventure games as each location can have its own individul menu options which can be updated depending on the player's actions.

Embedded menu commands
========================
Any menu string can optionally include a powerful set of embedded commands which allow you to customize the appearance of your menus to an incredible degree;. The list of commands in enclosed between sets of round brackets () and individual instructions Are separated using colons ":". For example:

```
        Pienu$(i) = " (Locate 10,10 s Ink 1,1) Hello"
```

Each instruction consists of just two characters which can be in either upper or lower case. Anything else will be ingnored completely. Most commands also require you to input one or more commands., These numbers fcmust never* make use of expressions « -these ars >•,-,+ &^:t..>« I.<x<J . TK= commands are listed below.

  Note: In the syntax the two important characters which make up the command are in upper case and highlighted bold.

BOB (draw a bob)

8 0 b n

The BOB command draws a bob number n at the current cursor position. No accound is taken of the hot spot of the bob,, All coordinates are measured relative to the top left corner,, Also note that colour zero is usually treated as transparent. This may bs changed using the NOMASK command from AMOS Basic, For examples

```
        Load "AMOS_DATA:Sprites/Octopus.abk11
        Henu*(l)="(Bob 1) 1":Menu*(l,1)="(Bob 2) 2"
        Menu$(l?2)="(Bob 3) 3"
        Menu On s Wait Key
```

ICON (draw an icon)

ICon n

Draws icon $ n at the current cursor position,. Note that unlike bobs,, colour zero is NOT normally transparent. See the Basic HAKE ICON MASK for more details*

LOCATE (move the graphics cursor)

LOcate x,,y

Tis command moves the graphics cursor to coordinates x,y measured relative to the top left corner of the menu line,, Note that after an instruction the graphics cursor will always be positioned at the bottom right of the object which has just been drawn. These coordinates will also be used to determine the location of any further items in your menu like so;

```
        Menu$(l)="Example "sMenu$(I,I)="Locate (Lo 50,50) in action "
        Menu$(i,2)="Guess my coords"
        Menu On : Wait Key
```

INK (set Ink and Paper colours)

INk n,, value

The INK command assigns the colour indexes to be used for the PEN, PAPER and OUTLINE colours, Here's a list of the various possibilitiess

n	Effect
1	Set text PEN colour
2	Set PAPER colour
3	Set OUTLINE colour

SFOMT (set font)

SFont n

SFont sets the current font to *graphics* font number n. This will be used in all future menu items. NOte that you MUST call GET FONTS before this instruction is executed, otherwise it can only use the two rom fonts. See EXAMPLE 16.5.

SSTYLE (set font style)

SStyle n

This coutmand sets the style of the current font to n which is a bit-pattern in the following format:

Bit	Effect
0	Underline
1	Bold
2	Italic

LINE: (draw a line) 224

LIne *x,y*

The LINE command draws a line from the current cursor position to the graphics coordinates x „y,. See EXAMPLE *16.,6*

SLIME (set line pattern)

SLine p

Sets the line style used in all subsequent LINE comands to the bit pa11ern held in p. Since there is no expession eva1uation, this pattern •should always be converted into decimal notation before use,, A simple demonstration of the possible line styles can be found in EXAMPLE 16.,7,,

BAR (draw a bar)

BAr x,y

This draws a rectangular bar from the current cursor coordinates to x,y. See EXAMPLE *Ib'-B,*

OUTLINE! (enclose bar with an outline)

Outline flag

Draws a border in the current outline colour (ink 3) around all subsequent bars,, A value of one activates the border and 0 removes it,, The general structure of a menu procedure is;

```
        Procedure ITEM
          If DREGC2)
            X=DRE6()sY==DREG(l)
              ...draw the item,..
          Endif
          DREG(O)=8X
          DREG(1)=BY
        End Proc
```

The dimensions of the menu item as displayed on the screen are set using the coordinates BX and BY. These MUST he loaded into registers DO and Dl before leaving your procedure as they are needed to create the final menu bar.

While inside your procedure you can perform most AMOS instructions including other procedures. But some instructions Af& absolutely-forbidden! If you use these commands, you won't get an error message but. your A MI (3 A may crash unexpectedly!

* NEVER change the current screen inside a menu,
* Don't set or reset a screen zone
* Avoid using instructions such as WAIT, WAIT KEY, INPUT or INKEY*
* Disc operations Are absolutely forbidden!
t Any error trapping in your procedure will be ignored.

Used with caution, the PROC command ca.n procedure some mind-blowing effects. For a demonstration, load EXAMPLE 16., 10.

RESERVE (reserve a local data i\re& for a procedure)

REserve n

Reserves n bytes of memory for this menu item.. This area can be accessed from within your menu procedure using the address held in AREG(i). The data Ares, you have created is common to all the strings in the current menu object. It can be used to exchange parameters between the various procedures called by a menu item.

MENU CALLED (redraw a menu item continually)

MENU CALLED(,,)

Automatically redraws the selected menu item 50 times a second whenever it's displayed on the screen. It's usually used in conjunction with a menu procedure to generate animated menu items which change in front cf you.r eyes.

In order to make use of this function, you first need to define a menu procedure, using the principles outlined above. Then add a call to this procedure in the required title strings using an embedded MENU CALL. When the user displays the chosen item, your procedure will be repeatedly accessed by the menuing system.

Since your procedure will be called 50 times a second, it should obviously return back to the menu as quickly as possible. This will allow enough •time for the rest of the menu to be succesfuily updated.

Also note that your embedded procedure can safely animate your item using either bobs or sprites. However, as the menu items *are* NOT double buffered, your bobs *m&y* flicker slightly on the screen. So it may be better to use computed sprites for this purpose instead,, Another approach is to draw your display with the standard AMOS graphics commands. An example of this can be seen in EXAMPLE 16.11.

MENU ONCEl (turns off automatic redrawing)

MEUN 0NCE(5,)

Turns off the automatic updating system started using the MENU CALLED,

Alternative menu styles
=========================

Normally the titles of a menu *Are* displayed as a horizontal line and the options are arranged below it in a vertical menu bar. If you. want to create something a little unusual, you can change the format of each level of your menu using the following three instructions:

MENU LINE (display a menu
as a horizontal line of items)

MENU LINE level
MENU LINE(,,)

Displays the menu options at the requested level in the form of a horizontal line. This menu line starts from the left-hand corner of the first title *ami* stretches to the bottom right corner of the last,.

MENU LINE level

Defines the menu style of an entire level of your menu,. This sould only be called during your menu definitions„

MENU LINE (,,)

Normally one would only use the "level" version for this command. Setting individual items to Line and Bar can give bizarre results, but this may be useful for something i

MENU TL1NE (display a menu as a total line)

MENU TLINE level
MENU TLINE(,,)

Displays a section of the menu as a "total line" stretching from the *v&ry* left of the screen to the *very* right, The entire line will be drawn even when the rist item is in the middle of the screen.

"level" is a number ranging from 1 to 8 which specifies the part of the menu to be affected. This is the standard form of the instruction, andchouldb [illegible]
have no effect.

You can also change the appearance of a menu after it has. been vrated using a second form of this command. For example:,

```
        Menu Lined,1) s Rem Displays menu 1,, 1 as a line.
```

I1EMU BAR (display a section of the menu as a bar)

MENU BAR level ,   v.
MENU BAR(..,,)

This displays the selected menu items in the form of a vertical bar. The width of this bar is automatically set to the dimensions of the largest item in your menu.

"level" is a number which indicates which part of the current menu definition is to be affected. As a default this option is used for levels 2 to 8 in your menu. Note that this form of the MENU BAR instruction may only be used during your programs initialisation phase,,

(,,) is a list of parameters which allow you to change the style of your menus once they've been installed,, Here's an example of Menu Bar and Menu Tlines

```
 :      FLAG=0
        SETJ1AN
        Do
          If Choice and Choice(l)=2 And Choice(2)=l Then ALTER
        Loop
        Procedure SETJ1EN
          Menu$(l)=" Bar Demo " : Menu*(2)=" Select Below "
          Menu*(:L,i)"-" I do nothing! "
          Menu*(2,1)=" Yes, press on me! "
          Menu On
        End Proc
        F'rocedure ALTER
          Shared ALTER
          Menu Del
          If FLAG=Q Then Menu Bar 1 ;: Flag==l Else Menu Tline 1 s Flag=0
          SET.J1EN
        End Proc
```

MENU INACTIVE (turn off menu item)

MENU INACTIVE level
MENU INACTIVE?,,)

As its name suggests, MENU INACTIVE deactivates a series of options in your menu. Any subsequent attempts to select these items will be completely ignored, "level" allows you. to deactivate an entire section of the fnenu and you can also deactivate individual menu options with the parameters (,,). These indicate the precise position of your item in the current menu hierarchy.

Note that the menu items you've turned off with the instruction will be immediately replaced by the INACTIVE* string you specified during your original menu definition. If this was omitted, allV" unavailable menuoptionswi3.1beshowninita1ics., MENU ACTIVE (activate a menu itsm)

MENU ACTIVE level
MENU **ACTIVE*,,)**

Simply reverses the effect of a previous I1ENLJ INACTIVE command. After you've called this instruction, the selected options will automaticall; redisplayed using their original title strings„

Moveable menus
==============

Ail OS menus can be displayed at any point on the screen, Hen us can be moved either ex piicity by your program or directly by the user.

MENU MOVABLE (activate automatic menu movement)

MENU MOVABLE level
MENU MOVABLE(,,)

Informs the menuing system that the menu items at "level" may be moved directly by the user - this is the default.,

The second form of this command allows you to set the status of each individual item in the menu,, The parameters between the brackets can indicate any position in the menu hierarchy..

Any menu *m&y* be repositioned by moving the mouse pointer over the FIRST item in the menu and pressing the left mouse button. A rectangular box will now appear around the selected menu item, *And* this may be moved to nay point on the current screen. When you release the left button the menu will be redrawn at the new position along with all the associated menu items.

Note that this command does not allow you to change the arrangement of any items below this level., If you want to manipulate the individual menu options you'll need to use a sea pa rate MENU ITEM command.. See . EXAMPLE 16.12 for a demonstration of this system..

MENU STATIC (fix a menu into place) •

MENU STATIC level
MENU STATIC(,,)

Defines the menu at "level" to be immoveable by the user,, One problem with moveable menu, is that the amount of the memory they consume will change during the course of a program. If your menus *&r<s* particularly large, or if memory is running tight, this can cause real problems as a single careless action by the user will abort your program with an "out of memory" error. With the help of the MENU STATIC command you can avoid this difficulty completely.

MENU ITEM MOUABLE (»=^«
individual menu options)

MENU ITEM MOVABLE level
MENU I TEH MOVABLE (,,}

This command is similar to MENU MOVABLE except that it allows you to re-arragne the various options in a particular level,, So all the items in a menu bar may been individually reposi tinned by the user,,

Normally it's illefal to move the items outside the current menu bar, but this can be overridden using the MENU SEPARATE command.

In order for the menu items to be moveable, the WHOLE! menu bar must also be moveable. So if you fix the MENU into paIce with MENU STATIC,, this command wi11 have no effeet„Additiona11yy you can't move the first item in the menu bar as this will move the entire line. Another side effect is that moving the last menu item will permanently reduce the size of your menu bar., There *B.re* two possible solutions;

t Enclose your entire bar with a rectangular box like sos

```
        Menu$(i,i}=,.,"(Bar 40,,100)(Loc 0,0)"
```

Where MENU$(l,i) is the first item in your current bar.

* Set the last item into place with MENU ITEM STATIC,

MENU ITEM STATIC (static menu item)

MENU ITEM STATIC level
MENU ITEM STATIC(,,) V :

This command locks one or more menu items firmly into place and is the default setting.

MENU SEPARATE (separate a list of menu items)

MENU SEPARATE level
MENU SEPARATEC,,)

Tells AMOS to separate all the items in the current level. Each item in your menu istreated completely independently from the previous one. If you haven't defined a background string, each item will be offset by two pixels from the one above. This creates an attractive stepped effect which can be removed by editing the menu with the MENU Accessory,

The optional parameters to this instruction allow you to split a menu *bar* at any point in the line. Once you've? separated an item it will be affected by the MENU MOVABLE commands rather than ITEM instructions.

MENU LINKED (link up a set of menus)

MENU LINKED level
MENU LINKED(,,)

This links one OP- niof-o menu items together. It's the opposite oT the? MENU SEPARATE instruction.

=MENU X (return the graphical X coordinate
of an menu item)

x=MENU X(,,)

The MENU X function allows you to retrieve the position of a menu item relative to the previous option on the screen. You can use this information to implement powerful menus such as the one found in EXAMPLE 16.13.

=MENU Y (return the graphical Y coordinate
of a menu item)

x=MENU Y(,,)

Returns the Y coordinate of a menu option, note that all coordinates *Are* measured relative to the previous item., So this is NOT a standard screen coordinate!

Moving a menu within a program
==============================

MENU BASE (move the starting point of a menu)

HEWU BASE x,y

This command moves the starting point of the first level of your menus to the absolute csreen coordinates x,y. All subordatine menu items will be displayed at their curent positions relative to the top of your flieniu See EXAMPLE 16,14 for a demonstration of the MENU BASE command in action.

SET MENU (move a menu)

SET MENU (,,) TO x,y

Sets the coords of the top left corner of a menu item. These coordinates *a. re* measured relative to the previous level. The starting point for the entire menu (coords 0,0) may be set with the MENU BASE command.

All the ivels of the menu below your die mi wlil also be moved by this instruction. Their relative positons will be unchanged. Since x,,y can be negative numbers, it's possible to arrange the items in a menu bar in the form of a control panel - see EXAMPLE 16,, 15.

Displaying a menu at the cursor position
==

MENU MOUSE (display the menu under the mouse)

MENU HOUSE ON/OFF

The MENU MOUSE features automatically display all menus starting from the current position of the mouse cursor,, The mouse coordinates are added to the MENU BASE to get the final position, so it's possible to place the menu a fixed distance away from the mouse pointer if required. See EXAMPLE 5.6,, 16,, The Amiga's sound system is capable of generating stereo sound effects which would have been unheard of just a few years ago. The results ;?,re impressive even through your TV speaker,, but when you connect your Amiga to a Hi-Fi, the sounds can actually shake your roomi

As you would expect from AMOS,, we've come a long way since the humbe BEEP command. In fact, we've provided everything you need to incorporate mind-blowing sound effects in your own games. All the AMOS sound commands are performed independently of your Basic programs. So your soundtracks can be played continuously,, without affecting the quality of the game-play in the slightest,,

• Samples may be created using any of the available sampling cartridges and can be replayed with a simple SAMPLAY instruction,, Each sample can be output in a variety of speeds, and may be looped repeatedly. It's even possible to play a sample as a musical note,

Music can be converted over from a separate package such as SOMIX, SOUNDTRACKER or GilC. The **AMOS** Music system is intelligent and will automatically stop when a sound is played through the current channel, thus allowing you to effortlessly combine both samples and music in the same sound channel, without the risk of unwanted interference effects.

Each song can incorporate up to 256 separate instruments; the only limit to the number of songs is the amount of available memory. In order to keep the memory overhead down to an absolute minimum, all tunes *are* built up of a number of separate patterns. Once a pattern has been created, it can be accessed anywhere in your music using just a couple of bytes,, By defining just a few key patterns,, you can therefore create dozens of different tues without running short of memory.

The best thing about the AMOS music system however, is that it's expandable. The entire source code is supplied on the data disc for you to examine or change„ So you won't be left out in the co1d by any future developments on the Amiga's music scene.

Simple sound effects

We'll start off with a run down of the built-in sound effects included in AMOS Basic. These are the AMOS equivalent to the Amiga Basic BEEP. command.

BOOM (generate a noise sounding like an explosion)

BOOH

Kapow! You're dead! Use BOOM to add the appropriate stereo sound effect in your games. Traditionally this type of "White Noise" as been extremely difficult on the Amiga, but AMOS uses a clever interrupt system to create *A* realistic explosion effect. Examples;

```
Boom : Print "You're DEAD!"
```

SHOOT (create a noise 1 i ke a gurs f i ring)

SHOOT

This command generates a simple gunshot effect. Like BOOM, SHOOT does not halt your program in any way,, So if you're firing several successive shots, you may wish to add a small delay using WAIT.

```
      Shoot : Wait 6 5 Shoot : Print "You're DEAD!"
```

BELL (simple bell sound) 234

BELL [f]

BELL produces a pure tone with frequency f. f sets the pitch of the note, from 1 *(very* deep) to 96 *(yery* high).

Sound channels

The Amiga's hardware can effortlessly play up to four different sounds simultaneously. This allows you to add attractive harmonics to your sound effects.

Each sound can be output through one of four VOICES numbered from 0 to 3, You can think of these voices as a separate instruments which can independently play their own sequence of notes, samples or music. All four voices *Are* internally combined to generate the final sound you hear through your speaker system.

The AMOS sound instructions will happily play your sounds using any arrangement of voices you. like. All AMOS sound commands use a standard way of entering your voice settings. Each voice is assigned a particular bit in a VOICE parameter like so:

```
        Bit 0 -> Voice 0
        Bit 1 -> Voice 1
        Bit 2 -> Voice 2
        Bit 3 -> Voice 3
```

To activate the required voices, set the appropriate bits to i. Here's a list of common values to make things a little easier

Value	Voice used	Effect
15	0,1,2,3	Uses all four voices
9	0,3	These voices *Are* combined together and output to the left speaker.
8	3	
6	2,4	Played through the RIGHT speaker.
4	2	
2	1	
1	0	

In order to do justice to the resulting sound effects, you'll almost certainly need to connect your Amiga to a Hi-Fi system of some sort, Host TVs *Are* just not capable of reproducing the full range of sounds which can be generated by the Amiga's amazing hardware.

VOLUME (change the sound colume) 235

VOLUME Cv,ǰ intensity

VOLUME changes the volume of the sounds which are to be played through one or more sound channels.

"intensity" refers to the loudness of this sound,, It can normally range from 0 (silent) to 63 (maximum). As a default, the volume is set to the same intensity for all four of the available voices. THe new volume will be used for all future sound effects,, including music.

The v parameter lets you change the volume of each voice independently, v now indicates which combination of voices are to be regulated,, This second option is only used by the sound effects. It has no affect on any music you're playing. The voices are selected using a bit amp in the standard format,, with each bit representing state of a single sound channel. If the bit is set to 1, then the volume of this voice will be changed, otherwise it will be unaffected,, Examples:

```
        Volume S0001,63 s Boom 2 Wait 100
        Volume mio ,,1A : Boom :: Wait 50
        Play 40,0 : uait 30
        Volume 50 : PIay 40,0
```

Sampled sound

If you had to generate all the sound effects you need., directly inside your computer, you would be faced with *An* impossible task. In practive, it's often much easier to take a real sound from an external source, such as a tape recorder, and convert it into a list of numbers which can be held in your computer's memory.

Eacn number represents the volume of a particular sample of the sound. By rapidly playing these values back through the Amiga's sound chips, you can recreate a realistic impression of the original sound. This technique forms the basis of the sampled sound effects found in most modern computer games,,

If you want to create your own samples, you'll be forced to buy a separate piece of hardware known as a SAMPLER CARTRIDGE. Although these cartridges are fun, they're certainly not essentia. AMOS Basic is perfectly capable of playing any existing sound sample,, without the need for any expensive add-ons.

Currently there *&re* hunderds of sound effects available from the public domain (PI)),, covering the vast majority of the effects you'll need in your games. We've even included a selection of useful samples on the AMOS data disc for you to experiment with.

SAM PLAY (play a sound sample from the AMOS sample bank)

SAM PLAY s
SAM PLAY v,s
SAM PLAY v5s,f

The SAMP PLAY instruction plays a sampled sound straight through your loudspeaker system. All samples are normally stored in memory bank number 5, but this may be
•fri3,~ily ch*ingsd u^ing the ^AM BANK command.

s is the number of the sample bank which is to be played,, There's no limit of the samples you can store in a bank other than the available memory. If you want to use your own samples with this instruction;, you'll need to incorporate them into an AMOS memory bank,. Full details can be found towards the end of this section.

v is a bit-map containing a list of voices your sample will use. As usual, there's one bit for each possible voice. To play your samples through the required voice, simply set the relevant bit to 1.

f holds the playback speed of your sample,, measured in hertz,, This specifies the number of samples which *Are* to be played each second,, Typical sample speeds range from 4000, for noises such as explosions;, to 10000 for recognisable speech effects,, By changing the playback rate, you can freely adjust the pitch of your sound over a large range,, So a single sample can be used to generate dozens of different sounds.
Examples

```
Load "AMOS_DATA;Samples/Sample_,Demo,,abk"
For S=l To 11
  Locate 0,0 i ? "Playing sample ";S
  Sam Play S
  Locate 0,24 sCentre "<Hit a key to continue)" sWait Key :Cline
Next S
Wait Key
Sam Play 1,11 ; Wait 5 s Sam Play 2,11
Wait key
Sam Play 1,1,2000
Wait Key
Sam Play 1,1,15000
```

A further demonstration of this command can be found in EXAMPLE 17.1

SAM BANK (change the current bank)

SAM BANK n

Assigns a new memory bank to be used for your samples,, All future SAM PLAY instructions will now take their sounds directly from this bank.

It's possible to exploit this feature to hold several complete sets of samples alongide each other. You can then between these samples at any time, with just a simple call to the SAM BANK.

SAM RAW (play a sample from memory)

SAM RAW voice,address,length,frequency

Plays a raw sample stored anywhere in the Amiga's memory, "voice" is a bit-pattern in standard format which specifies the list of voices your sample is to use. Each bit in the pattern selects a single channel to be played (see sound channels).

"address" holds the address of your sample. Normally, this will refer to the inside of an existing AMOS memory bank, "length" contains the length of the sample you wish to play,, "frequency" indicates the sample speed to be used *far* tho playback fin samples per second or Hz). This *m&y be very* different to the rate at which the sample was originally recorded.

SAM RAW lets you play standard Amiga samples straight through your loudspeaker, without the need to create a special memory bank (see Creating a sample bank)., It's now your responsibility to manage your samples in memory,, and enter the sample parameters by hand,, SAM RAW is great for browsing through files from your disc collection. Use B 1.0AD to hold a file in a bank and then use SAP! RAW to play the data,, With luck you should come across some interesting sounds. Examples:

```
        Reserve As Work 10,55000
         Bload "Samples/Samples.abk",start(iG)
        Sam Raw 15,start(10,length(10},10000
```

SAN LOOP (repeat a sample)

SAMP LOOP ON/OFF

The SAM LOOP directive informs AilOS Basic that all subsequent samples *are* to be repeated continuously. Examples:

```
        Load "'AMOSJ}ATAsSamples/Saiiipledemo.,abk"
        Sam Loop On
        For S=i To II
          Locate 0,0 : Print "Playing sample ";;S
          Sam Play S
          Locate 0,24 s Centre "<Hit a key to continued-" sWait Key sCline
        Next S
        Sam Loop Off
```

This looping effect can be deactivated with a simple call to the SAM LOOP OFF command.

Creating a sample bank

If you're indenting to play your own samples using SAM PLAY, you'll first need to load them into a memory bank. This can be achieved with the SAMMAKER program supplied on the AMOS data disc.

,, On start-up, SAMMAKER presents you with a standard AMOS file selector. Enter the filename of the first sample to be stored in your new bank, and press RETURN. If AMOS can't find the sampling rate,, you'll be asked to enter it directly. It doesn't really matter if you make a mistake at this point, as you can safely replay your samples at any speed you like.

After a short delay, you'll be prompted for the next sample to be installed into the bank. When you've reached the end of your samples;, type SAVE at the file selector to save your samples onto the disc. You'll be automatically prompted for the destination filename of your new bank. This can now be entered into AMOS Basic using the LOAD command like so:

```
        Load "Sample.abk"
        Load "Sample.abk" „6 % Rem Loads safliple into bank U6,,
```

Music

The AMOS music system allows you to easily add an attractive backing track to your games. Music can be created from a variety of sources,, including 6MC, SOUNTRACKER or SONIX.

In order to convert these musics into the special AI1OS format,, you'll need to use one of the translation programs included on the AMOS data disc. GMC music should have been saved using the SAME DATA icon,, as this copies both the music and the instrument definitions into a single large data file.

MUSIC (play a piece of music)

MUSIC n

The AMOS MUSIC command starts a piece of music from the music bank (S3). This music will be played independently of your Basic program, without affecting it in the slightest.

Normally, it's possible to store several complete arrangements in the same bank. Each composition is assigned its own individual music number. The only exception to this rule is music created by GMC, which only allows you to place one song in the bank at a time., Example;

```
        Load "flUSIC/Husicdemo.abk"
        Music i
```

The AMOS music, system is intelligent,, and will automatically suspend your music for the duration of any subsequent sound effects on the current channel. When the sound has finished, your tune will be restarted from its'previous position. Up to three separate tunes can be started at a time. Each new music command stops the current song,, and pushes its status onto a stack. Once the song has concluded, the old music will commence from where it left off.

MUSIC STOP (stop a single section of music)

MUSIC STOP

Halts the current piece of music. If another music is active, it will be restarted immediately,,

MUSIC OFF (turn off all music)

MUSIC OFF

THe MUSIC OFF command deactivates your music completely. In order to restart it, you'll need to execute your original series of MUSIC instructions again from scratch.

TEMPO (change the speed of a sample of music)

TEMPO s

TEMP modifies the speed of any tune which is currently being played with the MUSIC command, s is the new speed, and c«n range from .1 (Vf3ry slow) to 100 *(ver.y* fast). Not all instruments are capable of playing at this maximum speed, however,, The practical limit is closer to 50. For a demonstration, place the AMOS data disc into the current drive and type;

```
        Load "AMOS_DATA:Music/Musicdemo.abk"
        Music 1
        Tempo 35
        Tempo 5
```

Mote that music created with GMC often contains labels which set the tempo directly inside the arrangement. These labels will override the tempo settings within AMOS Basic. So it's not advisable to use them in your own music,

MVOLUME (set the volume of a piece of music)

MVOLUME n

Changes the volume of the entire piece of music to intensity n (0-63).

VOICE (activate one or more voices of a piece of music)

VOICE mask

Activates one or more voices of the music independently. Usually each voice will contain its own separate melody which will combined through your speakers to generate the eventual music,

"mask" is a bit mask in the normal AMOS format which specifies which voices you wish to play,, Each bit represents the state of one voice in the music. If it's set to 1, the voice will be played, otherwise it will be totally unused.

```
        Load "AMOS_DATA:Music/Musicdemo.abk"
        Music 3
        For V=0 To 15
        Locate 0,0 : Print "Voice ";V
        Voice V
        Wait 100
        Next V
        Direct
        Voice %0001 : Rem Activate voice 0
        Voice X0010 : Rem               1
        Voice £1001 : Rem               3 and 0
        Voice SI Hi s Rem               4
```

=VUMETER(volumemeter)

s=VUMETER(v)

The VUMETER function tests voice v and returns the volume of the current note which is being played by your music, s is an intensity value between 0 and 63. v is the number of a single voice to be checked (0-3).

[illegible]
music! Load EXAMPLE I7»2 for a demonstration™

Mote there's also an A11AL version of this intruction which allows you to create realtime VU meters using interrupts,. See the section on the V'U comniand for more infontation,,

Playing a note

PLAY (play a note)

PLAY [voice,] pitch,delay

Plays a single note through the loudspeaker of your TV or Hi-Fi. "pitch" sets the tone aft his sound, ranginafrom 0 (low) to 96{high). Rather than just being an arbitrary number, each pitch is associated with one of the notes (A,,,B,C,D,E,F,G)., This can be seen from the following table.

	Octave							
	0	1	2	3	4	5	6	7
Note	Pitch							
C	1	13	25	37	49	61	[illegible]	85
C#	2	14	26	38	50	62	74	86
D	[illegible]	15	27	39	51	63	75	87
m	4	16	28	40	52	64	76	88
E	5	17	29	41	53	65	77	89
F	6	18	30	42	54	66	78	90
FH	7	19	31	43	55	67	79	91
6	8	20	[illegible]	44	56	68	80	92
OM	9	21	[illegible]	45	57	69	81	93
A	10	[illegible]	34	46	58	70	82	94
AM	11	23	35	47	59	71	83	95
B	12	24	36	48	60	72	84	96

It should be apparent that the notes go up in a cycle of 12,, This cycle is known as an octave..

The optional voice parameter allows you to play your notes through any combination of the Amiga's four voices. As usutal it's a bit-map in the formats

Bit	Voice
0	0
1	i
2	2
3	3

Setting a bit to a value of 1 plays the relevant voice, "delay" sets the length of the pause between the play command and the next Basic instruction. This allows you to play each note before preceding the next one.

A delay of zero starts a note and immediately jumps to the next Basic: instruction,, By playing several notes after another,, you can easily generate some attractive harmonic effects. Examples:

```
Play 1,40,0 : Play 2,50,0
Wait Key
Play 1,40,15 : Play 2,50,15
Do
   T=Rnd(96) : V=Rnd(15) : Play V,T,,3
Loop
```

PLAY is not limited to purs notes incidentally.. It's also possible to assign complex waveforms to the sound generator using the powerful WAVE and NOISE commands.

Waveforms and envelopes

SET WAVE (define a waveform)

SET WAVE wave,shape*

The SET WAVE instruction provides you with the ability to define your *very* own instruments for use with the AMOS Basic PLAY instruction. The sound of yur instrument depends on the shape of a waveform held in the Amiga's memory. This forms a template which is repeated to produce your final note.

"wave" is the number of the waveform you wish to define. Allowable wave numbers start from 2 onwards. That's because waves zero and 1 *are* already installed. Wave zero holds a random noise pattern for producing explosion effects. Wave one is a smooth sine wave and generates the pure tones used by the standard PLAY instruction.,

The shapes of your waveform *&re* set using a list of 256 numbers which are entered using the SHAPE$ parameter. Now look at the uppest diagram in the AM0S4.PIC (file included with this manual packet).

< picture AM0S4.PIC, the uppest diagram > 242

Each number represents the intensity of an individual section of the waveform. This is equivalent to the height of just one point in the diagram. Possible values for intensity range from -128 to 127. Since AMOS strings are only capable of holding ^positive* numbers (0-255),, you'll need to convert your negative values into a special internal format before use. The required value can be calculated by simply adding 256 to the negative numbers in your list,,

Here's a program which demonstrates how the triangular wave in the previous diagram could be created in AMOS Basic

```
S$=""
For I=-128 To 127
  X=I : If X<0 Then Add X,256
  S*=S$+Chr$(X)
Next I
Set Wave 2,S*
```

Before playing your waveform you have to tell AMOS Basic which channels *are* to be assigned to your wave. This can be achieved using the WAVE command. Add the following line to the previous routine

```
Wave 2 To IS s For 3=10 To 60 s Play S.,10 :: Next S
```

The Best way to reproduce the effect of a real instrument is to combine several SINE waves together. An example of one of these sine waves can be seen in the picture AM0S4.PIC:

< picture AII OS 4. PIC, the diagram in the middle > 243

Adding several of these waves together, with different sizes and separate starting points, produces waves in the •following pattern:

< picture AH0S4.,PIC., the lowest diagram >

This generates the smooth harmonics needed for your notes,, Here's an example:

```
SHAPE$^'" 5 Degree
For S=0 To 255
  V=Int((Sin(S)/2+SIN(S*2+45)/4)*128)+127
  SHAPE$=SHAPE$+Chr$(V)
Next S
Set Wave 2,SHAPE* : Wave 2 to 15
For W-10 to 60 s Play hi, 10 : Next N
```

WAVE (assign a wave to one or more sound channels)

WAVE w To v

WAVE assigns wave number w to one or more sound channels, v contains a bit™map in the standard format. If a bit in the pattern is set to 1 then the approrpriate voices *are* used by PLAY., otherwise they will be completely unaffected. 244

As a default, wave zero is reserved for the NOISE channel, and wave one contains a sine wave. Here *&re* some examples:

```
Wave 0 To £0001
Play 1,40,0
Wave 0 To £1100
Play 20,10
Wave 1 To £1111
Play 60,0
```

NOISE (assign a noise wave to a channel)

NOISE TO voices

Applies a white noise effect (wave 0) to the selected voices,, Load EXAMPLE .17,3 for a demonstration.

"voices" is a standard bit pattern. The first four bits represent the four possible voices, starting from zero,. NOISE is equivalent to the command;

```
Wave 0 To voices
```

Exampless

```
Noise To 15
Play 60,0
Play 30,0
```

DEL. WAVE (delete a wave)

DEL USAUE n

Deletes a wave which has previously been defined using SET WAVE,, n is the number of the wave, and starts at 2. It's impossible to delete the built-in NOISE ans SINE waves using this instruction,, After the wave has been erased, all voices will be reset to the standard SINE wave (default).

SAMPLE (assign a sample to a wave)

SAMPLE n TO voices

This is the most powerful cersion of all the wave commands. It assigns a sample stored in the sample bank to the current wave. Play will now, take an instrument straight from the sample bank,

```
        Load "Samples/sample!,,abk"                                2 4 5
        Sample 1 To 15
        For 1=20 To SO
          Play I.,50
        Next 1
```

As usual "voices" allows you to select a range of voices to be set by the instruction. It's a standard bit-map;; Bit 0 ==> Voice 0   etc...

Motels The range of notes that a sample can be played with, depends on its original recording rate,, If a no to is too high, AH OS may not be able to play it at all. The acceptable range varies from a sample to sample,, but it's usually between 10 and 50,,

SET ENVEL (create a volume envelope)

SET ENVEL wave,phase TO duration,volume

The SET ENVEL command smoothly changes the volume of a note while it's being played. In the real world, sounds don't just sprint into existence fully formed. They tend to evolve over a period of time, according to a pattern known as the volume envelope. The shape of this envelope varies depending on the type of instrument you *are* playing. A typical example of one of these envelopes is shown in the picture AM0S5.PIC.

< picture AMOS5.PIC >

The sound is split, up into four phases? Attack decay., sustain and release. AJ1QS E<asic allows you to define your envelopes using up to seven separate steps. Each step represents a steady change in the volume of the current note,, 246

"wave" is a number of the waveform which will be affected by this instruction. It's possible to use any waveform you like for this purpose, including the built-in NOISE and SINE generators.

"phase" holds the number of the particular phase which is to be defined, ranging from 0 to 6,,

"duration" specifies the length of the current step in units of a 50th of a second™ This determines the apparent speed of the volume change to be generated in this phase,,

"volume" specifies the volume which is to be reached by the end of this phase. Allowable volume levels range from 0-63,, It's important to understand that this volume is relative to the intensity you've previously st with the VOLUME command. So even if the note is quiet,, the shape of the envelope will be perfectly reserved. Now for some examples!

```
        Set Envel 1,0 To 200,63 : Rem Sets the 1st step.
        Play 40,0
```

As you can hear, the volume of your sound starts from zero, and increases to a maximum intensity during the length of the note. Now let's try defining something a little more complicated,,

```
        Set Envel 1,0 To 15,60
        Play 40,0 s Wait Key
        Set Envel 1,1 To 1,50
        Play 40,0 s Wait Key
        Set Envel 1,2 To 10,50
        Play 40,0 : Wait Key
        Set Envel 1,3 To 50,0
        Play 40,0
```

Finally, here's an example of a NOISE envelope:

```
        Noise To 15
        Set envel 0,0 To 1000,30
        Play 40,0
        Wait Key
        Music Off
```

Don't confuse waves and envelopes. A wave sets the frequency components of your notes, whereas an envelope simply changes their volume according to a set pattern.

Speech

Your Amiga is supplied with a powerful speech synthesizer program which *CAP,* be found on the standard Workbench disc, With the help of this routine, your AMOS programs can be made to speak. Speech is especially userful in education, as many yound people will respond far better to the spoken word than to boring text.

One word of caution though. Since the narrator package is independent of AMOS Basic, we can't attest to its absolute reliability. You're unlikely to encounter any serious problems, but it's well worth treating it with a littlle care.

SAY (speak a phrase)

SAY t*C,,mode;i

The SAY command is incredibly easy to use. Enter your text in normal English, concluding your phrase with a punctuation mark such as full stop. SAY will now translate your words into an internal format and speak them directly through your loudspeaker,, Example:

```
        Say "AMOS Basic can really speak"
```

The first time you use this instruction,, the narrator ..device will automatically be loaded from disc:,. So it's vital to ensure that an appropriate disc is placed in the current drive before using this system,, as otherwise you may get an Intuition style requester box,

"mode" toqgeles between two separate speech modes. As a default, your program will wait for the duration of the speech,, and any music or sound effects will be temporarily suspended. Setting "mode" to a value of one activates multitasking system which allows you to output your speech whilst AMOS is executing your program. Inevitably,, this will slow down your basic routines Considerably, To return your speech back to normal, set mode to zero,,

If the narrator system cannot understand what you are attempting to speak you won't get an error message]., but the command will be automatically aborted. Also note that the narrator can occasionally get slightly confused with *very* short sentences. Sometimes the remainder of the previous phrase is tagged to the end of the current voice. The problem can be solved by simply adding a list of spaces to the end of your text. These will wipe out the unwanted speech data. "

SET TALK (set speech effects)

SET TALK sex,,mode,,pitch,,rate

This allows you to change the type of voice which will be used by the SAY command, "sex" chooses between a male (0) or female (1). In all honesty, it's not a particularly realistic rendition. Better effects can be created by simply increasing the frequency of the voice using the pitch parameter.

"mode" adds a strange rhythmic pattern to the voice. This can be activated by setting "mode" to a value of 1.

"pitch" changes the frequency of the voice,, from 65 to 320.

"rate" specifies the speed,, measured in the words per minute (40-400),,

Any of the above parameters can be omitted if required. Providing you keep the commas in their normal positions,, you can change *Any* set of options independently.

Filter effects
==============

LED (activate a high pass filter/
change power led)

LED ON/OFF

The LED command has two completely separate actions,, Not only does it toggle the POWER led on your Amiga's console (in KickStart versions 1.3 just makes the led a little darker)., but it also controls a special high pass filter.

The filter changes the way high frequency sounds are treated by the system. Normally,, these sounds are filtered out so as to avoid the risk of unwanted distorion effects. Unfortunately, this robs many percussion instruments of their timbre,, By turning off the filter, you can recapture the essential quality of many instruments.

AMDS Basic provides you with dozens of useful keyboard commands,, These can be used in anything from an Arcade game to an Adventure™ It's, even possible to write a fully fledged wordp.rocessor entirely in AMOS Basic!

=1NKEY$ (function to get a keypress)

k*=INKEY$

This function checks whether the user has pressed a key., and returns its value in the string !<$•

Mote the IMKEY$ command doesn't wait your input in any way,, If the user hasn't entered a character, INKEY* will simply return an empty string "",,

1MKEY$ is only capable of reading keys which return a specific Ascii character from the keyboard. Ascii is a standard code used to represent ail the characters which can be printed on the screen,,

It's important to realise that some keys,, like HELP button or the •function keys,, use a rather different format,, If INKEY$ detects such a key,itwi11retu.rnac:haracterwithava1ueofiero(CHR$(()))„Youcan now find the internal scan code of this key using a separate SCAM CODE function.

=SCANC0DE (input the scancode of the last key input with IMKEY*)

s=SCANCODE

SCANCODE returns the internal scancode of a key which has previously en tered using the INKEY$ f unction » This a 11 ows yau to c.heck f oi" keys which do not produce a character from the keyboard, such as HELP or TAB. Type the following small examples

```
Do
  While. K$~""
    K$=Inkey$
  Wend
  If Asc(K*)-=0 Then Print "You pressed a key with no ASCII Code"
  Print "The Scancode Is";Scancode
  K$=""
Loop
```

=KEY STATE (test whether an individual key has been pressed)

t^KEY STATE(s)

Check if a specific button has been pressed on the Amiga's keyboard, s is the internal scancode of the key you want to check. If this key is currently being depressed then KEY state will return a value of true (-1), otherwise the result will be false (0).

=KEY SHIFT (return the status of the shift keys)

keys=KEY SHIFT

KEY SHIFT returns the current status of the various control keys,, These keys such as SHIFT or Alt cannot be detected using the standard INKYf- or SCANCODE system. But you can easily test for any combination of control keys with just a single call to the KEY SHIFT function,, "keys" is a hit map in the following format;

Bit	Key Tested	Notes
0	Left SHIFT	
1	Right SHIFT	
2	Caps Lock	Either ON or OFF
3	CTRL	
4	Left ALT	
5	Right ALT	
6	Left AMIGA	C= key on some keyboards
?	Right AMIGA	

If a bit is set to a one,, then the associated button has been held down by the user.

IKIPUT$(n) (function to input n characters into a string)

INPUTS enters n characters straight from the keyboard, waiting for each one in turn. As with INKEYt, these characters a.re not schoed onto the screen.

x$ is a string variable which will be loaded with your new characters, n holds the number of characters to be entered. Examples

```
Clear Key s Print "Type In Then Characters"
C$=INPUT*(10) s Print "You entered ";C*
```

This insturction Knot* the same as the standard INPUT command,, The two instuctions are completely different,, Also note that there's a special version of INPUT* which can be used to read your characters from the disc,

WAIT KEY(wait for a keypress)

WAIT KEY

Waits for a single keypress.

KEYSPEED(change key repeat speed)

KEY SPEED lag,speed

KEY SPEED lets you tailor the speed of the keyboard to your own particular taste. The new speed will be used for every part of the AMOS system, including the editor,

"lag" is the time in 50th of a second between pressing a key, and the start of the repeat sequence.

"sDeed" is the delay of second between each successive character.

CLEAR KEY (initialise keyboard buffer)

CLEAR KEY

Whomever you enter a character from the keyboard;, its Ascii code is placed in an area of memory known as the keyboard buffer- It is this buffer that is sampled by the INKEY* function to get your key presses.

CLEAR key erases this buffer completely, and returns your keyboard to this original state. It's especially helpful at the start of a program, as the buffer may well be full of unwanted information. You can also call it immediately before a WAIT KEY comand to ensure that the program waits for a fresh key press before preceding.

PUT KEY (Put a string into the keyboard buffer)

PUT KEY a$

Loads a string a characters directly into the keyboard buffer,, Carriage returns can be included using a CHR$(13) character.

The most common use of PUT KEY is to set up defaults for your input routines,, Here's a demonstration:

```
Do
  Put Key "Wo"
  Input "Another Game'1:; A $
  If A$="No" Then Exit'
Loop
```

Input/Output

INPUT (load a value from the user and put it a variable)

INPUT

Provides you with a standard way of entering information into one or more variables. There *are* two possible formats for this instrucion:

INPUT vars[;]

Enters a list of variables directly from the keyboard., "var" can contain any set of variables you like,, separated by commas,. A question mark will be automatically displayed at the current cursor position.

INPUT "Prompt";variable list[;3

Prints out the "prompt" string before entaring your information. Note that you must always place a semi-colon between your text and the variable list. You are *not* allowed to use a comma for this purpose,.

The optional semi-colon ";" at the end of your variable list specifies that the text cursor will not be affected by the INPUT instruction, and will retain its original position after the data has been entered»

When you execute one of these commands,, Basic will wait for you enter the required information from the keyboard,, Each variable in your list must be matched by a single value from the user,, These values must be of the same as your original values, and should be separated by commas„

LINE INPUT (input a list of variables separated by a Return)

LINE INPUT "Prompt" ^variable].ist[;]

Line input is exactly same as INPUT, except that it uses a Return instead of a comma to separate each value you enter from the keyboard.,

PRINT / ?

(print a list of variables to the screen)

PRINT items *

The PRINT instruction displays some information on the screen,, starting from the current cursor position.

Each element in your list must be separated by either semi-colon or a comma. A semi-colon prints the data immediately after the previous value, whereas a comma first moves the cursor to the next TAB position on the screen.

Normally the cursor will be advanced downwards by a single line after each PRINT instruction. This can be suppressed by adding a **separator** after the print.

```
PRINT 10,20*i0,"Hel"5
PRINT "lp"
```

USING (formatted output)

PRINT USING format*5variable list

The USING statement is used in conjunction with PRINT to provied fine control over **the** format of your printed output.

format$ specifies a list of characters which defines the *way* your variables will be displayed on the screen. Any normal text in this string will be printed directly, but if you include one of the characters '"S+-.;" then one of a range of useful formatting operations will be performed.

"" Formats a sting variable,. Every "" is replaced by a single character from your output string, taken from left to right.

```
PRINT USING "This is a w"~- demonstration of USING";"Small"
```

8 Each hash character specifies a single digit to be printed out from your cariable,. Any unused digits in this list will be automatically replaced by spaces. 255

+ Adds a plus sign to a number if its positive, and a minus minus sign if it's negative,

```
PRINT USING "+«tt";10 5 PRINT USING "+«it";-10
```

- Only includes a sign if the number is negative,, Positive numbers are preceded by a space,,

• Places a decimal point in the number,, and centres it neatly on the screen„ . ?

; Centres a number but doesn't output a decimal point.

^ Prints out a number in exponential form,.

```
PRINT USING "Here is a number ^"; 12345.678
```

REM / ' (comment)

REM comment

The REM statements is'used to add comments to your Basic program.. Any text typed in after a REM statement will be completely ignored by AMOS Basic.

```
REM This is a comment
' this is a comment.
```

So, a quote mark "'" can also be used, but it *must* be placed at the absolute beginning of the line.

DATA (place a list of data items in a AMOS Basic program) 256

The DATA statement allows you to incorporate whole lists of useful information directly inside a Basic program. This data can be subsequently loaded into one or more variables using the READ instruction. Each variable in your list is separated tay a single comma.

```
DATA 1,2,3,"Hello"
```

Unlike most other Basics, the AMOS version of this instruction also lets you include expression s as part of your data. So the following lines of code a.re equally acceptable:

```
DATA $FF50,,$890
DATA miiilillili,,^1101010101
DATA A
Labels Data A+3/2.0-Sin(B)
Data "Hello" +••"There"
```

It's important to realise that the "A" at LABEL will be input as the contents of variable A, and not the character A., The expression will be evaluated automatically during the READ operation using the 3. as test values of A and B.

Also note that each DATA instruction must be the only statement on the current 1ine. Anything after this command will be totally ignored! Data statements can be placed anywhere in your Basic program. However., any data you store inside an AMOS procedure will not be accessible from the main program„

READ (read some data a DATA statement into a variable)

READ list of variables

Loads some informatoin stored in a DATA statement into a list of variables. READ uses a special marker to determine the location of the next piece of data to be entered. At the start of your program, the marker is moved to the first item of the first DATA statement. Once this item has been read, the marker is advanced so that it points to the next item in your list,, As you might expect, the variables you read f

must be exactly the same type as the data held at the current position,
Elxamples

```
        T=10
        Read A$,B,C,D$
        Print At,B,C,D*
        Data "String",2,T*20+rnd(100),"AMOS '-i-"3asic:"
```

RESTORE: (set the current READ pointer) 257

RESTORE Label
RESTORE LABEL*
RESTORE Line
RESTORE number

RESTORE changes the point at which a subsequent READ operation will expect to find the next DATA statement. Each AMOS procedure has its own individual data pointer. So any calls to this command wlil only apply to the ^current* procedure!

"label" is a label which specifies the position of the first DATA statement to be read. This label name can be calculated as part of an expression. The following Basic commands are perfectly legals

```
        RESTORE L…
        RESTORE "L"+"A"+"e"+"E"+"L"
```

Similarly, line selects the line number of the next DATA statement-Like "label" it can be entered as an expressions

```
        RESTORE TEST+2
```

By allowing you to jump at will through the DATA statements in your program, RESTORE lets you choose your information depending on the actions of the user,, Each room of an adventure,, for instance, could have its description stored in a list of simple DATA statements,, To read this description you could use some thing likes

```
        Restore R00M*5+1.000 :: Rem Each R00I1 has 5 data statements
        Read DESC$ s Print DESC$
```

Obviously, if a data statement does not exist at the line specified by RESTORE, and appropriate error message will be generated. Beware of trying to use this command inside a procedure,, In order to work, your DATA statements #I1UST# be within the current procedure.

WAIT (wait in SOths of a second) 258

WAIT n

Suspends an AMOS Basic program for n/50 of a second,, Any functions which use interrupts, such as WOVE and MUSIC, will continue to work as normal during this period,,

=TIHER= (count in SOths of a second)

v=TIMER
TIMER=v

TIMER is a reserved variable which is incremented by 1 every 50th of a second. It's commonly used to set the seed of the random number generator like so:

```
        Randomize Timer
```

NOT (logical NOT operation)

v=NOT(d)

This function changes every binary digit in a number from a 1 to a 0 and vice versa. Since True=-1 (%1111111111111111) in binary and False=0, NOT(True)=False. Example:

```
        Print Bin$(Not(%1010),4)
( results 0101 )
        If Not(True)=False Then Print "False"
```

TRUE (logical TRUE)

v=TRUE

Whenever a test is made such as X>10, a value is produced. If the condition is true then this number is set to -1, otherwise it will be zero.

```
     If -1 Then Print "Minus 1. Is TRUE"
     If TURE Then Print "and TRUE Is ";TRUE
```

FALSE (logical FALSE)

v=FALSE

Returns a value of zero. This is used by all the conditional operations such as IF...THEN and REPEAT...UNTIL to represent FALSE.

```
        Print FALSE
( results 0 )
```