# Menus

In this Chapter, the AOZ programmer will learn how to create, control and use powerful on-screen menus. These techniques allow you to customise your own menu designs and operations, and offer true interactivity.

AOZ menus can have as many as eight overlaid levels and any menu item can be repositioned anywhere on screen. There is no restriction to the inclusion of title styles and graphic images, and your own Bobs and icons can be used directly.

When reading your menus, branching to user-selected points in your programs can be automatic, whether triggered by the mouse or directly from the keyboard. And if you cannot wait to see all this in action, the Chapter is accompanied by a full range of ready-made demonstration programs available on the AOZ Tutorial pack.

## Using AOZ menus

NOTE: work in progress, not all the options will be implemented. TODO: correct once done...

*SELECTING*

All of these menus are activated by holding down the right mouse button. Once the relevant menu has
appeared on screen, drag the mouse cursor over the option you wish to select and release the button. The selected option number is automatically returned to your program.

*REPOSITIONING*

A menu can be repositioned on screen by placing the mouse cursor over its lop left-hand corner and holding down the left mouse button. When a small box appears on the menu bar, drag it across the screen using the mouse. To freeze the current position of a menu, hold down the [Shift] key as well. This allows you to explore the menu without activating any of its options.

AOZ menus can be created directly from within your programs, or you may prefer to use the menu
defining program supplied on disc.

## Simple menus

## *MENU$*

**reserved variable**: Define a menu title or option

```
Menu$(number)=title$
Menu$(number,option)=option$
```

To create a simple menu, its title line must first be defined. Each heading in a title line created with MENU$ must be assigned its own number. The title at the left-hand edge of the title line is represented by 1, the next title by 2, and so on, from left to right. The characters in your title string hold the name of the numbered title. This example sets up a menu title line offering two

titles, and you should note the use of the spaces to separate titles when they appear in the title line:

```
Menu$(1)," Action"
Menu$(2)," Mouse"
```

The second type of usage of MENU$ defines a set of options that will be displayed in the vertical menu bar. The brackets after MENU$ contain two parameters, the first is the number of the menu heading that your option is to be displayed beneath, followed by the option number you want to install in the vertical menu bar. All options are numbered downwards from the top of the menu, starting from 1. The option string holds the name of your new option, and can consist of any text you choose. The following lines could be added to the last example above:

```
Rem Action menu has one option
Menu$(1,1)=" Quit " : Rem Ensure three spaces after Quit
Rem Mouse menu has three options
Menu$(2,1)="Arrow " : Rem Ensure five spaces after Arrow
Menu$(2,2)="Cross-hair"
Menu$(2,3)="Clock " : Rem Ensure five spaces after Clock
```

That specifies your list of alternatives for the "Action" and "Mouse" menus. Before this program can be run, it must first be activated.

## *MENU ON*

**instruction**: Activate a menu

```
Menu On
```

Use this command to initialise the menu previously defined by a MENU$, and the menu line will appear when the right mouse button is pressed. To activate the previous example, add the following lines:

```
Menu On
Wait Key
```

Trigger the menu and its options now, and use the left mouse button to re-locate the title bar. Now that this simple menu has been activated, the selected options must be read and reported back to the system.

# Reading a simple menu

## *CHOICE*

**function**: Read a menu

```
selection=Choice
title number=Choice(1)
option number=Choice(2)
```

CHOICE will return a value of True if the menu has been highlighted by the user, otherwise a value of False is returned. After the status of your menu is tested, the value held by CHOICE is automatically re-set to zero.

CHOICE(1) will return the value of the title number which has been chosen.

CHOICE(2) will return the value of the option number which has been selected.

Now remove the Wait Key from the last example, and replace it with the following lines. This should change the shape of the mouse cursor, depending on the option selected from your menu. Note that Choice=-1 can be simplified to Choice.

```
Do
    If Choice and Choice(1)=1 Then Exit
    If Choice(1)=2 and Choice(2)<>0 Then Change Mouse Choice(2)
Loop
```

# Creating advanced menus

The use of MENU$ and CHOICE is not limited to the creation of simple menus. In fact, their use can be extremely sophisticated.

MENU$ is used to define the appearance of each individual item in one of your menus, whether it is a title, an option, a sub-option, all the way down to the eighth layer of options in the menu hierarchy. In this Chapter, when "single item parameters" is used it simply means those numbers separated by commas and held inside a single pair of brackets, that refer to the position of a single item somewhere in the menu. Up to eight parameters can be used, separated by commas. To be clear, here are some examples of parameters defining the position of a single item in the menu hierarchy:

```
Menu$(1)="Title1"
Menu$(1 ,1)="Title1 Option1"
Menu$(2,3)="Title2, Option2"
Menu$(1,1,1,1)="Title1, Option1, Sub-option1, Sub-sub-option1"
```

Now look at these uses of MENU$, which are used to give a single item its own characteristics:

## *MENU$*

**instruction**: Define appearance of a single item in a menu

```
Menu$(single item parameters)=normal$
Menu$(single item parameters)=normal$,selected$,inactive$,background$
```

Normal$ is simply the string of characters that make up the normal appearance of an item when it is displayed on screen. The following strings are all optional.

The selected$ changes the appearance of the item when it is selected by the mouse. As a default, selected items are highlighted by printing the string in inverse text.

The inactive$ comes into effect when an item has been deactivated using the MENU INACTIVE command, which
is explained later. It can be used to display an alternative text or appearance, but if it is omitted, inactive items are automatically displayed in italics.

The background$ creates a background effect for menu items when they are initially drawn, such as a box or a border created by the internal BAR or line drawing commands.

Similarly, the CHOICE function can return the option selected at a required level in the menu hierarchy. For example:

```
Menu$(1)="Title"
Menu$(1,1)="Option 1"
Menu$(1,2)="Option 2"
Menu$(1,2,1)="Option 2.1"
Menu On
Do
    If Choice Then Print Choice(1),Choice(2),Choice(3)
Loop
```

For very large menus, the IF structure as used in the last example would become unwieldy, and cause delays while the menus were being read. AOZ provides a method for handling the largest of menus.

## *ON MENU PROC*

**instruction**: Automatic menu selection

```
On Menu Proc procedure1
On Menu Proc procedure1,procedure2
```

Each title in your menu can be assigned its own procedure which will be executed automatically when that option is selected by the user. Like the other ON MENU commands that are described next, ON MENU PROC uses interrupts, which means that it is performed 50 times a second. So your program can be engaged in other tasks while the menus are continually checked by the system.

When automatic selection takes place as the result of ON MENU PROC, the procedure is executed and the program will be returned to the instruction immediately after the ON MENU call. Procedures can make use of the CHOICE function to monitor which option has been triggered, and to perform the appropriate action.

## *ON MENU GOSUB*

**instruction**: Automatic menu selection

```
On Menu Gosub label1
On Menu Gosub label1,label2
```

Depending on which option has been selected by the user, ON MENU GOSUB goes to the appropriate subroutine. After using this instruction, ON MENU should be used to activate the menu system before jumping back to the main program with a RETURN. Also note that the labels used with this command cannot be replaced by expressions, because the label will be evaluated once only when the program is run.

## *ON MENU GOTO*

**instruction**: Automatic menu selection

```
On Menu Goto label1
On Menu Goto label1, labe12...
```

Although this command is available for use, it has been superseded by the more powerful ON MENU PROC and
ON MENU GOSUB instructions. It is retained to provide compatibility with programs written in STOS Basic.

## *ON MENU ON/OFF*

**instruction**: Toggle automatic menu selection

```
On Menu On
On Menu Off
```

To activate the automatic menu system created by the ON MENU PROC, GOSUB or GOTO commands, simply
give the ON MENU ON command. After a subroutine has been accessed in this way, the system is automatically disabled. Therefore you must reactivate the system with ON MENU ON before returning to the main program.

To suspend the automatic menu system, ON MENU OFF is used. This can be vital if your program is executing a procedure which must be performed without interruptions, such as loading and saving information to disc. Menus are reactivated using ON MENU ON.

## *ON MENU DEL*

**instruction**: Delete labels and procedures used by ON MENU

```
On Menu Del
```

Use ON MENU DEL to erase the internal list of labels or procedures created by the range of ON MENU
commands. You are warned that this command can only be used after menus have been deactivated by ON MENU
OFF.

# The Menu control commands

## *MENU ON*

**instruction**: Activate a menu

```
Menu On
Menu On bank number
```

The simple form of this command has already been dealt with at the beginning of this Chapter. After MENU ON, a menu is displayed when the user next presses the right mouse button. If an optional bank number is included after the command, the appropriate menu will be taken from the numbered memory bank. Please see MAKE MENU BANK, below.

## *MENU OFF*

**instruction**: Deactivate a menu

```
Menu Off
```

This command temporarily turns a menu off, making it inactive. The menu can be reactivated at any time with the MENU ON command.

## *MENU DEL*

**instruction**: Delete one or more menu items

```
Menu Del
Menu Del(single item parameters)
```

On its own, MENU DEL erases the whole menu. But be warned, once the menu has been deleted it CANNOT be
retrieved!

MENU DEL can also be qualified by up to eight parameters, separated by commas, and held in a single pair of brackets. These values represent the precise position of the item in the menu hierarchy to be deleted. For example:

```
Menu Del(1) : Rem Delete title number 1
Menu Del(1,2) : Rem Delete option 2 of title 1
Menu Del(2,3,4) : Rem Delete sub-option 4 of option 3 of title 2
```

## *MENU TO BANK*

**instruction**: Save menu definitions into a memory bank

```
Menu To Bank number
```

Use this command to save your menu along with its entire structure of branch definitions to the numbered bank. Once the menu has been stored in the selected memory bank, it will automatically be saved along with your Basic program. By storing your menu definitions in a memory bank, the size of your program listings are reduced significantly, freeing valuable space in the editor memory. If the bank number you select already exists, the appropriate error message will be given.

## *BANK TO MENU*

**instruction**: Restore a menu definition saved in a menu bank

```
Bank To Menu number
```

Follow BANK TO MENU with the number of the memory bank where your menu data is stored. The menu will be
restored to its exact state when originally saved, so the restoration process may take a few seconds. To activate the restored menu, call MENU ON.

## *MENU CALC*

**instruction**: Recalculate a menu

```
Menu Calc
```

Any item in an AOZ menu can be changed during the course of a program. This is extremely useful
for designing adventure games or creating self help programs, where individual menu options can be updated depending on the user's actions. After the menu has been defined, items may be added and options replaced as you please, and everything will be repositioned automatically as soon as the menu is called up with the right mouse button.

This process may take a few seconds, particularly with very large menus. The MENU CALC command is
designed to perform this recalculation at the most suitable point in the program, in order to minimise any delays.

You are advised to freeze your menus with MENU OFF at the start of the recalculation procedure, to prevent the user calling the menu half way through an update. It may then be made active again using MENU ON after the updating process is over.

# Alternative menu styles

The AOZ programmer is free to change the display format of any level of any menu, and design a customised layout. As a default, all titles are displayed in a horizontal bar with their related options arranged below in a vertical block. Here are the alternatives:

## *MENU LINE*

**instruction**: Display menu options as a horizontal line

```
Menu Line level number
Menu Line (single item parameters)
```

Use this command to change the display of options that relate to a particular title from a vertical block into a horizontal line. The line of options will now start from the left-hand corner of the first menu title and stretch to the bottom right-hand corner of the last title. Follow MENU LINE with the number of the level you want to affect, and make sure that this command is only called during your menu definitions. The level number can range from 1 to 8, and it specifies the layer of the menu to be affected.

It is perfectly legal to set individual items by this method, and with the following MENU TLINE and MENU BAR commands. This can result in some highly eccentric displays.

```
Menu Line(1,1,1) : Rem Display sub-option 1,1,1 as a line
```

## *MENU TLINE*

**instruction**: Display a menu as a total line

```
Menu Tline level number
Menu Tline(single item parameters)
```

MENU TUNE is used to display a section of your menu as a total line, stretching from the extreme left to the extreme right of the screen. The complete line will be drawn even if the first item is centre screen. Use this instruction in the same way as MENU LINE during your menu definitions.

## *MENU BAR*

**instruction** : Display menu items as a vertical bar

```
Menu Bar level number
Menu Bar(single item parameters)
```

This instruction displays the selected menu items as a vertical bar whose width is automatically set to the length of the largest item in the menu. As a default, this option is used for levels 2 to 8 of your menu, and it must be used during the program's initialisation. There will be no effect if it is called after the menu has been activated.

When followed by a list of bracketed parameters, MENU BAR can also be used to change the style of your menus once they have been installed. Here is an example of a customised menu layout:

```
FLAG=0
SET_MEN
Do
    If Choice and Choice(1)=2 and Choice(2)=1 Then CHANGE
Loop
Procedure SET_MEN
    Menu$(1)="Try me first " : Menu$(2)="Select me " : Rem Four spaces
    Menu$(1,1)="1 am useless " : Rem Five spaces
    Menu$(2,1)="Please select me!"
    Menu On
End Proc
Procedure CHANGE
    Shared FLAG
    Menu Del
    If FLAG=0 Then Menu Bar 1: FLAG=1 Else Menu Tline 1: Flag=0
    SET_MEN
End Proc
```

## *MENU INACTIVE*

**instruction**: Turn off a menu item

```
Menu Inactive level number
Menu Inactive(single item parameters)
```

Use this command to turn off options in your menu. By selecting the number of a level from 1 to 8, all items in that level will be deactivated. If you define an individual item in brackets by giving its parameters, only that item will become inactive.

If no inactive string has been defined when you originally set your menu up with MENU$, any menu options that have been made inactive will be shown in italics. Otherwise the special inactive string will appear.

## *MENU ACTIVE*

**instruction**: Activate a menu item

```
Menu Active level number
Menu Active(single item parameters)
```

MENU ACTIVE reverses the effect of a previous MENU INACTIVE command. An entire level or single item
specified by its parameters can be re-activated and the original appearance of their title strings will be re-displayed.

# Moving menu displays

As has been explained, AOZ menus can be displayed anywhere on your screen. The display positions can be moved either by the user or by your program.

## *MENU MOVABLE*

**instruction**: Activate automatic menu movement

```
Menu Movable level number
Menu Movable(single item parameters)
```

The default condition is that the menu items at a particular level may be moved directly by the user. Any level can be repositioned by moving the mouse pointer over the first item in the menu and holding down the left mouse button. A rectangular box will appear around the selected menu item, and it can be dragged to its new screen position. When the left mouse button is released, the menu is re-drawn at this location, along with all of its associated items.

Use MENU MOVABLE to set the status of entire menu levels, or selected items in a menu hierarchy, but please note that this command does not allow you to change the status of any items below the selected level.

## *MENU STATIC*

**instruction**: Fix a menu in static position

```
Menu Static level number
Menu Static(single item parameters)
```

One characteristic of mobile menus is that the amount of memory they use changes during the course of the program. With large menus or programs that are on the boundary of available memory this can cause real problems.

MENU STATIC can be used to avoid these difficulties by setting the level or item at which the entire menu becomes immovable by the user.

## *MENU ITEM STATIC*

**instruction**: Fix items in static positions

```
Menu Item Static level number
Menu Item Static(single item parameters)
```

This command locks one or more menu items into place, and is the default setting.

## *MENU ITEM MOVABLE*

**instruction**: Move individual menu options

```
Menu Item Movable level number
Menu Item Movable(single item parameters)
```

This is similar to MENU MOVABLE, but it allows the re-arrangement of various options in a particular level. Normally it is not possible to move items outside of the current menu bar, but this can be overcome by the MENU SEPARATE command, which is explained below.

To use MENU ITEM MOVABLE for changing the position of a menu item, the entire menu bar itself, must be
movable. So if MENU STATIC has been called, this command will have no effect. The first item in a menu bar can not be moved, because this would move the entire line. Furthermore, if the last item in a menu bar is moved, the size of that bar will be permanently reduced.

This problem can be overcome either by setting the last item into place with a MENU ITEM STATIC command, or by enclosing the whole menu bar with a rectangular box, like this:

```
Menu$(1 ,1)=,,,,"(Bar40,100)(Loc0,0)"
```

## *MENU SEPARATE*

**instruction**: Separate a list of menu items

```
Menu Separate level number
Menu Separate(single item parameters)
```

This command is used to separate all the items in the numbered level of the menu. Each item will then be treated independently. If no background string has been defined, every item is offset from the preceding item by two pixels, creating a stepped effect, which can be removed by editing from the Menu utility.

By specifying the parameters of a single item after the MENU SEPARATE command, a menu bar can be split at any chosen point. Once an item has been separated it can be affected by the MENU MOVABLE command instead of the ITEM instructions.

## *MENU LINK*

**instruction**: Link a list of menu items

```
Menu Link level number
Menu Link(single item parameters)
```

This is the exact opposite of MENU SEPARATE, and is used to link one or more items together.

## *X MENU*

**function**: Return the graphical x-coordinate of a menu item

```
x=X Menu(single item parameters)
```

The X MENU function allows you to get the position of a menu item, relative to the previous option on screen. This information can be used to set up very powerful menus.

## *Y MENU*

**function**: Return the graphical y-coordinate of a menu item

```
y=Y Menu(single item parameters)
```

Y MENU returns the y-coordinate of a menu option, measured relatively to the previous item on screen.

Please refer to the demonstration program above.

# Moving a menu within a program

## *MENU BASE*

**instruction**: Move the starting position of a menu

```
Menu Base x,y
```

Use this command to move the starting point of the first level in your menu hierarchy to the absolute screen coordinates at x,y. All subordinate menu items will now be displayed relative to this starting point.

## *SET MENU*

**instruction**: Move a menu item

```
Set Menu(single item parameters) To x,y
```

SET MENU sets the screen position of the top left-hand corner of the menu item whose parameters are given in brackets. These coordinates are measured relative to the previous level, so the starting point for the entire menu can be set by the MENU BASE command. All levels of the menu below this single item will also be moved by your SET MENU command. The coordinates can be negative as well as positive, so you are free to position items anywhere on screen.

## *MENU MOUSE ON / MENU MOUSE OFF*

**instruction**: Display the menu at position of mouse cursor

```
Menu Mouse On
Menu Mouse Off
```

Use these commands to toggle the display of all menus starting from the current position of the mouse cursor. The mouse coordinates are added to the MENU BASE to calculate the menu position, so it is possible to lace a menu at a fixed distance from the mouse pointer.

# Keyboard shortcuts

Menus are an extremely useful system of selecting from a clear choice of options. They present the user with a simple method of performing some complex operations, and they are particularly suitable for the less experienced or younger user. But the AOZ programmer can be more concerned with speed rather than simplicity, and menu operations can become a little tedious. This is why you may prefer to choose your options directly from the keyboard.

AOZ allows you to assign a keyboard shortcut to any of your menu items, and these key presses are
interpreted as their exact equivalents. They can be used with any menu command, including the ON MENU range.

## *MENU KEY*

**instruction**: Assign a key to a menu item

```
Menu Key(single item parameters) To c$
Menu Key(single item parameter) To scancode, bitmap
```

Any key can be assigned to an item in a previously defined menu, provided that the item specified is at the bottom level of the menu. In other words, keyboard shortcuts cannot be used to select sub-menus because each command must correspond to a single option in the menu.

In its simplest form, define the single item parameters as usual, by giving their hierarchy numbers in brackets after MENU KEY. Then assign the item TO a string containing a single character. Any additional characters in this string will be ignored.

TODO - "Update from Amiga definitions below, to include AOZ"

Because each key on the Amiga keyboard is assigned its own scancode, this code can be made use of for those keys that have no Ascii equivalents, the so-called control keys. Here is a simple routine to print out scancodes:

```
Do
    Repeat
        A$=inkey$
    Until A$<>""
    Z=Scancode
    Print Z
Loop
```

The following scancodes can also be used with the MENU KEY command, instead of a character string:

# Scancode Keys

- 80 to 89 Function keys [Fl] to [F10]
- 95 [Help]
- 69 [Esc]

An optional bitmap can also be added, to check for control key combinations such as [Ctrl] + [A]. Here are the alternatives:

Bit - Key - Tested Notes

- 0 - left [Shift] - only one [Shift] key can be tested at a time
- 1 - right [Shift] - only one [Shift] key can be tested at a time
- 2 - [Caps Lock] - either ON or OFF
- 3 - [Ctrl] -
- 4 - left [Alt] -
- 5 - right [Alt] - this is the [Commodore] key on some keyboards
- 6 - left [Amiga] -
- 7 - right [Amiga] -

If more than a single bit is set in this pattern, several keys must be pressed at the same time in order to call up the associated menu item. Any of these keyboard shortcuts can be erased by using MENU KEY with no parameters. For example:

```
Menu Key(1,10) : Rem Erase shortcut assigned to item (1,10)
```

Here is an example that checks for key presses of the Amiga's ten function keys:

```
Menu$(1)="Function Keys"
For A=1 To 10
    OPT$="F"+Str$(A)+" "
    Menu$(1,A)=OPT$
    Menu Key(1,A) To 79+A
Next A
Menu On
Do
    If Choice Then Print "You have pressed Function Key ";Choice(2)
Loop
```

# Embedded menu commands

AOZ menus offer complete freedom to make use of any text styles of graphics you want. The final part of this Chapter deals with the commands that make this possible.

Any menu string can include a powerful set of optional embedded commands that allow you to customise the
appearance of your menus. These embedded commands must be enclosed between sets of brackets, and individual commands must be separated by colons, like this:

```
Menu$(1)"(LOcate 10,10: Ink 1,1)I am embedded"
```

Each embedded command consists of only two characters, which can be in either upper or lower case. Any other characters will be ignored. So the following characters will be treated as "LO" when entered as an embedded command:

```
LO
lo
locate
Lonniedonegan
```

Most embedded commands also require you to input one or more numbers. These numbers must never make use of expressions, because they will not be evaluated.

In the listings for all of the following embedded commands, the two important characters that make up the command are in upper case bold type.

## *LOCATE*

**embedded command**: Move the graphics cursor

```
LOcate x,y
```

The LOcate embedded command moves the graphics cursor to coordinates x,y measured relative to the top left-hand corner of the menu line. Please note that after this command, 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. For example:

```
Menu$(1)="Example " : Menu$(1,1)="Locate (LO 50,50) in action"
Menu$(1,2)="Please guess my coords"
Menu On : Wait Key
```

## *BOB*

**embedded command**: Draw a bob

```
BOb number
```

The BOb command draws the specified Bob image from the Object Bank at the current cursor position. The
existence of any hot spot will be ignored. Colour zero will normally be treated as transparent, but this can be changed with NO MASK. All coordinates will be measured relative to the top left-hand corner.

## *ICON*

**embedded command**: Draw an icon

```
ICon number
```

ICon draws the given icon number at the current cursor position. Colour zero is not normally transparent in this case, but transparency can be achieved with MAKE ICON MASK, as detailed in Chapter

## *INK*

**embedded command**: Set pen, paper or outline colour

```
INk mode,value
```

The INk command assigns the colour index values to be used for the pen, paper and outline colours in your menu drawing. The numbers to be used for the various modes are as follows:

Number - Mode

- 1 - Set text PEN colour
- 2 - Set PAPER colour
- 3 - Set OUTLINE colour

## *SFONT*

**embedded command**: Set font

```
SFont number
```

SFont sets the current menu font to the selected graphics font number. This font will now be used for all subsequent menu items. GET FONTS must be called before this instruction is executed.

## *SSTYLE*

**embedded command**: Set font style

```
SStyle bit-pattern
```

SStyle sets the style of the "Current font to the selected bit-pattern. In the following table, a setting of 1 will have the listed effect, whereas a setting of zero will have no effect:

```
Bit Effect
0 underline
1 bold
2 italic
```

## *LINE*

**embedded command**: Draw a line

```
LIne x,y
```

LIne draws a line from the current cursor position to the graphics coordinates x,y.

## *SLINE*

**embedded command**: Set line pattern

```
SLine pattern
```

SLine sets the line style to be used in all subsequent LIne commands to the selected bit-pattern. Because there is no evaluation of expressions, the bit-pattern must be converted into decimal notation before use.

## *BAR*

**embedded command**: Draw a bar

```
BAr x,y
```

BAr draws a rectangular bar from the current cursor coordinates to x,y.

## *PATTERN*

**embedded command**: Draw a pattern

```
PAttern number
```

PAttern changes the fill pattern used by the BAr command to the numbered style.

## *OUTLINE*

**embedded command**: Enclose a bar with an outline

```
OUtline value
```

OUtline draws a border in the current outline colour (set to ink colour 3) around all subsequent bars. A value of 1 activates the border and a value of 0 removes it.

## *ELLIPSE*

**embedded command**: Draw an ellipse

```
ELlipse radius1 ,radius2
```

ELlipse draws an ellipse centred on the current coordinates, with the chosen radii. To draw a circle centred at the current coordinates, simply make radius1 equal to radius2.

## *PROC*

**embedded command**: Call a procedure
TODO - "Not sure if this whole section below needs an added update to include AOZ too?"

```
PRoc NAME
```

PRoc allows you to call any AOZ procedure directly within a menu line. The called procedure cannot include any parameters, otherwise a syntax error will be generated.
This is the command that allows you to customise your menu to your own needs and ignore the limitations of the available menu commands.

At the start of your procedure, the following values are held in the Amiga's 68000 processor registers:

- DREG(0) holds the graphical x-coordinate of the top left-hand corner of the current menu item. Do not draw graphics to the left of this point on the screen unless you want to confuse the menu re-drawing process and generate bizarre effects.
- DREG(1) holds the y-coordinate of your menu item. Avoid drawing below this point on the screen to minimalise
  possible errors.
- DREG(2) holds the current status of your menu drawing operations. It contains a value of 0 (false) while the menu item is being drawn, in which case you must load DREG(0) and DREG(1) with the x,y-coordinates of the bottom right-hand corner of your menu zone, and return from the procedure immediately.

  If DREG(2) is -1 (true), you are free to perform the graphics operations used by the procedure. After completion, you should return the coordinates of the bottom right-hand corner of your item in DREG(0) and DREG(1) as above.
- DREG(3) holds a value of -1 if the menu is selected and the first menu string is on display, otherwise it will contain a value of 0.
- DREG(4) is set to TRUE when the menu branch is initially opened.
- AREG(1) holds the address of the zone created with RESERVE. It is used to allow different procedures to communicate with one another.

Here is the general structure of a menu procedure:

```
Procedure ITEM
    If DREG(2)
        X=DREG(0) : Y=DREG(1)
        drawing instructions go here
    Endif
    DREG(0)=BX : Rem x coord of bottom right corner of menu item
    DREG(1)=BY : Rem y coord of bottom right corner of menu item
Endproc
```

The dimensions of the menu item as it is displayed on screen are set using the coordinates BX and BY. These must be loaded into registers DREG(0) and DREG(1) before leaving your procedure because they are needed to create the final menu bar.

While inside your procedure, most AOZ instructions can be performed, including other procedures.

However, the following rules must be observed to avoid your computer crashing!

- Never change the current screen inside a menu.
- Do not set or re-set a screen zone.
- Avoid instructions that halt the action of your program (WAIT, INPUT, INKEY$, etc)
- Errors will bypass any error trapping in the procedure, and the program will return to the editor after closing the procedure.

## *RESERVE*

**embedded command**: Reserve a local data area for a procedure

```
REserve number of bytes
```

REserve allocates the chosen number of bytes of memory for a menu item. This area can then be accessed from within your menu procedure using the address held in AREG(1). The data area that is reserved in this way is for the storage of variables. This area is local to the menu item that calls the procedure.

# Automatic re-drawing of menus

The last two commands in this chapter affect the automatic process which re-draws the selected menu 50 times every second.

## *MENU CALLED*

**instruction**: Re-draw a menu item continually

```
Menu Called(single item parameters)
```

MENU CALLED engages the automatic re-drawing process. This command is normally used with a menu
procedure to generate animated menu items, often with spectacular moving graphic effects.

To use this facility, a menu procedure should first be defined, as explained above. Next, add a call to this procedure in the required title strings, using an embedded PRoc command. Finally, activate the updating process with MENU CALLED. When the user selects the chosen item, your procedure is repeatedly accessed by the menu system.

Because menu items are not double buffered, bobs may flicker a little, but the use of computed sprites will present no such problems.

## *MENU ONCE*

**instruction**: Turn off automatic re-drawing

```
Menu Once(single item parameters)
```

MENU ONCE turns off the automatic updating system instigated by MENU CALLED. After the command is given, each menu item will only be re-drawn once when the menu is called on the screen. It is used like this:

```
Menu Once(1,1)
```