The Joystick and Mouse

This Chapter clarifies all aspects of controlling and exploiting the joystick and mouse in your programs.

Gamepads

AMOS Professional on the Amiga only supported "joysticks". AOZ on the contrary, supports modern controllers with triggers, sticks and multiple buttons.

GAMEPAD CONNECTED

function: returns TRUE if the gamepad is connected to the computer

```
PADNUMBER = 0
CONNECTED = Gamepad connected( PADNUMBER )
If Gamepad Connected( 0 )
    Print "Press button A to play!"
End If
```

PADNUMBER is an integer which value from 1 to 4, you can connect up to 4 gamepads to the machine...

GAMEPAD BUTTON

function: return TRUE if a given button of a gamepad is pressed.

```
PADNUMBER = 0
BUTTONNUMBER = 1
Do
    DOWN = Gamepad Button( PADNUMBER, BUTTONNUMBER )
    If DOWN Then Print "You have pressed button B!"
Loop
```

BUTTONNUMBER is an integer number from 0 to 15, with the following mapping of the buttons:

```
- 0 : Bottom button in right cluster ('A' on XBox controller, used by the Fire
function)
- 1 : Right button in right cluster ( 'B' on XBox controller)
- 2 : Left button in right cluster ('X' on XBox controller)
- 3 : Top button in right cluster ('Y' on XBox controller)
- 4 : Top left front button
- 5 : Top right front button
- 6 : Bottom left front button
- 7 : Bottom right front button
- 8 : Left button in center cluster (gamepad left 'control' button)
- 9 : Right button in center cluster (gamepad right 'control' button)
- 10 : Left stick pressed button
- 11 : Right stick pressed button
- 12 : Top button in left cluster (JUp)
```

```
- 13 : Bottom button in left cluster (JDown)
- 14 : Left button in left cluster (JLeft)
- 15 : Right button in left cluster (JRight)
```

GAMEPAD AXE

```
PADNUMBER = 0
AXISNUMBER = 2
Do
    VALUE# = Gamepad Axe( PADNUMBER, AXISNUMBER )
    Print "Position of the horizontal axis of the right stick : "; VALUE#
Loop
```

AXISNUMBER is an integer number from 0 to 3, indicating which axe to report. The function returns a floating point value from -1 to 1.

```
- 0 : Horizontal axis for left stick
- 1 : Vertical axis for left stick
- 2 : Horizontal axis for right stick
- 3 : Vertical axis for right stick
```

GAMEPAD TRIGGER

```
PADNUMBER = 0
TRIGGERNUMBER = 0
Do
    VALUE# = Gamepad Axe( PADNUMBER, TRIGGERNUMBER )
    Print "Position of the left trigger stick : "; VALUE#
Loop
```

TRIGGERNUMBER is an integer number from 0 to 1, indicating which trigger to report. The function returns a floating point value from 0 to 1.

```
- 0 : Left trigger
- 1 : Right trigger
```

GAMEPAD NAME$

function: return the name of the gamepad as a string.

```
Do
    If Gamepad Connected( 0 )
        Print "You gamepad is a : "; Gamepad Name( 0 )
    End If
    Wait Vbl
Loop
```

Gamepad mapping

HTML5 joystick APIs beging rather new, a large number of gamepads (specially the cheqap ones) do not respect the specification chart, and some or all the buttons, axis or trigger do not share the same numbers as XBOX gamepads.

This is the reason why AOZ allows you to remap the number of the buttons with the Gamepad Map instructions...

TODO: put instructions back and continue...
TODO: Modernise text for Joysticks. Remove Amiga conventions/number of ports etc

Joysticks

A joystick can be used to control movement around the screen by pushing its handle in the desired direction, and to trigger all sorts of actions by pressing one or more buttons built in to its mechanism. Either of the two joystick sockets at the back or side of your Amiga will happily accept a joystick plug. If two users want to control one joystick each for specially written programs, both ports can be used. To make a joystick interact with your programs, the computer needs to be able to read its movements and actions.

JOY

function: read status of joystick

```
status=Joy(port number)
```

This inspects what is happening with the joystick and makes a report. If the joystick you are interested in is plugged into the joystick port, the computer must be told to look at port number (1). If you are using the mouse port call that port number (0). For example:

```
Do
    J=Joy(1)
    Print Bin$(J,5),J
Loop
```

When you run that routine, reports are given about the movements of the joystick and the status of the fire-button in the form of binary numbers. The pattern of ones and zeros in the report can then be inspected. Binary bits shown as zero indicate that nothing is happening, whereas if any of the bits in the report is shown as a one, it means that the joystick has been moved in the direction that relates to that bit. Here is a list of those bits along with their meanings.

Bit number Meaning

- 0 Joystick has been moved Up
- 1 Joystick has been moved Down
- 2 Joystick has been moved Left
- 3 Joystick has been moved Right
- 4 Fire-button has been pressed

Each of those aspects of the joystick status can be accessed individually, using the following functions:

JLEFT

function: test for joystick movement towards the left

```
x=Jleft(port number)
```

This returns a value of True if the joystick connected to the given port number has been pushed to the
left, otherwise a value of False.

The three other function in this family are self-evident, as follows:

JRIGHT

function: test for joystick movement towards the right

```
x=Jright(port number)
```

JUP

function: test for joystick movement upwards

```
x=Jup(port number)
```

JDOWN

function: test for joystick movement downwards

```
x=Jdown(port number)
```

These functions can be demonstrated by the following example:

```
Do
    If Jleft(1) Then Print "WEST"
    If Jright(1) Then Print "EAST"
    If Jup(1) Then Print "NORTH"
    If Jdown(1) Then Print "SOUTH"
Loop
```

FIRE

function: test status of fire-button

```
x=Fire(port number)
```

To set up a routine for testing to see if the fire-button has been pressed, use the FIRE function followed by the joystick port number. A value of True will be given only if the fire-button on the relevant joystick has been pressed.

```
Do
    F=Fire(1)
    If F=-1 Then Centre "BANG!": Shoot
    Print
Loop
```

The mouse pointer

The mouse is often used in practical programming whereas joysticks have become associated with playing computer games, but they both do much the same thing. They can both control moving objects on screen and be used to select from a range of on-screen options, using a cursor.

The mouse cursor has been pre-programmed to look like a pointer arrow, along with two additional standard shapes that can be selected at any time. The standard shapes have been assigned the numbers one to three, as follows:

- 1 Arrow pointer (default shape)
- 2 Cross-hair
- 3 Clock

CHANGE MOUSE

instruction: change the shape of the mouse pointer

```
Change Mouse number
```

To change the shape of the pointer arrow, use this command followed by the number of the required shape listed above. For example:

```
Do
    For N=1 To 3
        Change Mouse N
        Wait 25
    Next N
Loop
```

There is no need to restrict your choice to these three shapes. If you select an image number greater than three, AOZ will look at an image stored in the sprite bank, and install it as the mouse pointer. The first image in the bank may be called up by using Change Mouse 4, the second by specifying number 5, and so on.

HIDE

instruction: remove the mouse pointer from the screen

```
Hide
Hide On
```

This instruction hides the mouse pointer by making it invisible. Although it cannot be seen, it is still active and sending back reports, and the position of the mouse pointer co-ordinates can still be read. AOZ will automatically count the number of times that the HIDE instruction is used, and employ this number to SHOW the mouse pointer once again at your command. If you prefer to keep the mouse pointer invisible all the time and ignore the counting system, use the special ON version of the instruction, like this:

```
Hide On
```

SHOW

instruction: reveal the mouse pointer back on screen

```
Show
Show On
```

This makes the mouse pointer visible again after a HIDE instruction.

As a default, the system counts the number of times that the HIDE command has been used, then reveals the pointer on screen when the number of SHOWs equals the number of HIDEs. To bypass this counting system and reveal the mouse pointer immediately, use SHOW ON.

```
Do
    For N=1 To 10
        Hide : Wait N : Show
    Next N
Loop
```

Reading the status of the mouse

Whether or not the mouse pointer is visible, the computer must know two things in order to make any use of the mouse. It needs to recognise where the mouse pointer is as well as if any of the mouse buttons have been pressed.

X MOUSE

_reserved variable: report or set the x-co-ordinate of the mouse pointer

```
X Mouse
x=X Mouse
```

X MOUSE reports the current location of the x-coordinate of the mouse pointer. Because movement is controlled by the mouse rather than by software, coordinates are given in hardware notation, which is demonstrated by the following example:

```
Do
    Print X Mouse
Loop
```

This can also be used to set a new coordinate position for the mouse pointer and move it to a specific position on the screen. This is done by assigning a value to X MOUSE as if it was a Basic variable. For example:

```
For N=200 To 350
    X Mouse=N
    Print X Mouse
Next N
```

Y MOUSE

reserved variable: report or set the y-coordinate of the mouse pointer

```
Y Mouse
y=Y Mouse
```

Y MOUSE is used to give the y-coordinate of the mouse pointer in hardware co-ordinates, or to reposition the mouse pointer on screen, and it is employed in exactly the same way as X MOUSE.

```
For N=150 To 300
    X Mouse=N : Y Mouse=N/2
    Print X Mouse : Print Y Mouse
Next N
```

MOUSE KEY

function: read status of mouse buttons

```
k=Mouse Key
```

The MOUSE KEY function checks whether one of the mouse buttons has been pressed and makes a report in the form of a binary pattern made up of these elements:

- Bit 0 Left mouse button
- Bit 1 Right mouse button
- Bit 2 Third mouse button if it exists

As usual, the numbers zero and one make up the report, with a one displayed when the relevant button is held down, otherwise a zero is shown. Try this routine:

```
Curs Off
Do
    Locate 0,0
    M= Mouse Key : Print "Bit Pattern ";Bin$(M,8);" Number ";M
Loop
```

MOUSE CLICK

function: check for click of mouse button

```
c=Mouse Click
```

This is similar to MOUSE KEY, but instead of checking to see whether or not a mouse button is held down,
MOUSE CLICK is only interested in whether the user has just made a single click on a mouse button. It returns the familiar bit pattern of these elements:

- Bit 1 Single test for left mouse button
- Bit 2 Single test for right mouse button
- Bit 3 Single test for third mouse button, if available

These bits are automatically re-set to zero after one test has been made, so they will only check for a single key press at a time. Here is an example:

```
Curs Off
Do
    M=Mouse Click
    If M<>0 Then Print "Bit Pattern ";Bin$(M,8);" Number";M
Loop
```

Limiting the mouse pointer

One of the commonest screen conventions for both leisure and serious programs is the use of control panels. AOZ relies on them extensively for ease of use and clarity. Supposing you need to set up a control panel on your screen, but you want to prevent the mouse pointer from wandering outside the area of that panel.

LIMIT MOUSE

instruction: limit mouse pointer to part of the screen

```
Limit Mouse x1 ,y1 To x2,y2
Limit Mouse
```

This command sets up a rectangular area for the mouse pointer to move around, and traps it inside the boundaries ,set by hardware coordinates, from the rectangle's top-left TO bottom right-hand corner. For example:

```
Limit Mouse 300,100 To 350,150
```

If you need to restore freedom to the mouse pointer and allow it to move around the entire screen, use the LIMIT MOUSE instruction on its own, without any coordinates after it. Note that SCREEN OPEN must be followed by a WAIT VBL command before LIMIT MOUSE can be used, otherwise no screen will be set up for screen limits to be set.

Finding the mouse pointer

If you already understand the concept of different screens and screen zone numbers, you will appreciate that it is not difficult to lose track of the mouse pointer. You may need to keep a check on various screens and screen zones in order to keep in control of the mouse pointer.

If you do not already understand the concept of different screens and screen zone numbers, you will need to become familiar with the various SCREEN commands and ZONE functions.

MOUSE ZONE

function: check if the mouse pointer is in a zone

```
zone number=Mouse Zone
```

The MOUSE ZONE function checks to see where the mouse pointer is currently located, and if it has entered a screen zone, the number of that zone is returned. It is equivalent to the following line:

```
X=Hzone(X Mouse,Y Mouse)
```

MOUSE SCREEN

function: check which screen the mouse pointer is occupying

```
screen number=Mouse Screen
```

Use MOUSE SCREEN to return the number of the screen where the mouse pointer is currently located, like this:

```
X=Mouse Screen
Print X
```

Displaying menus with the mouse pointer

Finally, as an AOZ programmer, you will want to make use of the automatic facility for displaying all the menus whose root starts from the current position of the mouse pointer.

MENU MOUSE

instruction: display menu under current mouse pointer location

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

When this facility is turned ON, any menus that have been set up at the current location of the mouse pointer are instantly displayed on screen. The mouse coordinates are added to the MENU BASE in order to calculate the position where the menus are displayed, so you are able to place a menu at a pre-set distance away from the mouse pointer if you like. To stop this automatic process, simply use MENU MOUSE OFF.

Please see Chapter XX.X (6.5) which deals with all aspects of AOZ menus, if you are not experienced in their use.