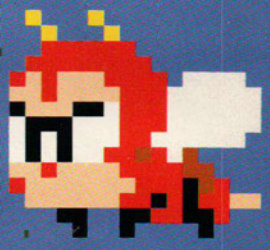
BRITAIN'S BEST-SELLING MAGAZINE FOR AMIGA OWNERS

AMIGA FORMAT

ISSUE 10 / MAY 1990 / £2.95

WORK HARD

Your essential guide to getting down to business



COVER 10 DISK

TWO GREAT PLAYABLE DEMOS

VIVID IMAGE'S ACTION PACKED HAMMERFIST ➤

HOVERBOARD ACTION IN WIPE OUT FROM GONZO GAMES

NO AMIGA COVERDISK?
DEMAND ONE FROM YOUR NEWSAGENT NOW!

PLAY HARD

Is Rainbow Islands the game of the year?

REVIEWS: COLOUR VIDI ■ CLASS OF THE '90S ■ ULTRADESIGN TUTORIALS: DIGITISING ■ GAMES PROGRAMMING ■ COLOUR GRAPHICS

9 770957 486004 05

THE WHOLE TRUTH ABOUT games programming PART 4

aliens

DAVE JONES, the ace coder behind Psygnosis' hits *Menace* and *Blood Money*, spills more secrets.

In this and next month's articles I will detail the largest part of the *Menace* code: namely, the alien movement and control routines. There will be no source with this month's instalment because I will only be describing WHAT the routines accomplish – next month will cover HOW they are accomplished, along with the source.

Movement is Life

So, we want to have aliens flying about the screen in nice patterns, attacking your ship, launching missiles at you, tracking your movements and doing all the other things that aliens do best. All the basic ingredients of a good shoot-em-up. What should spring to mind to control all of the movements is a decent data structure. As I have said before, there is no substitute for sitting down with a pen and paper and having a good think about what has to be accomplished. The best idea is to start off with the basic data that would have to be stored about an alien flying about on screen. This would probably be the following:

Alien number
Alien X coordinate
Alien Y coordinate
Alien speed

There will also have to be some way of telling the alien what to do. The simplest form would be a coordinate for it to move to. Once it gets there it would need another coordinate to go to. It would do this repeatedly until we tell it to stop, or until is has been destroyed. First, then, we will define the basic structure to describe a particular alien

```
            rsreset         ; reset the rs counter
X.Pos       rs.w    1       ; one word for the alien's x coord
Y.Pos       rs.w    1       ; one word for the alien's y coord
Sprite.Num  rs.b    1       ; a byte to hold the alien number
Speed       rs.b    1       ; a byte to hold the alien speed
```

Following this in memory would be a table of x,y coordinates for the alien tc move to in sequence. An example piece of source that moves an alien from left to right and back again could be defined as follows:

```
dc.w=   0,100                   ; start alien at x=0, y=100
dc.b    6,2                     ; use alien number 6 with a speed of 2

dc.w    320,100,0,100,$7fff     ; move to 320,100 then to 0,100
                                ; the $7fff is an end marker
```

This basic structure will provide only limited control of an alien, so it's time to add a few more features. Most of the aliens are animated (Figure One, overleaf, shows an eight-animation *Menace* alien) so as well as defining the alien number, we also have to define its current animation number. Note that in games an animated object is usually given a number that relates simply to the base address of that alien. The number of animations it actually has is stored separately. This makes changing the number of animation frames very simple, because the alien numbers all remain exactly the same. So we will also have to store the number of animations the alien contains somewhere in the structure as well as its current animation number:

```
Anim.Num    rs.b    1       ; a byte to hold the current animation
Num.Anims   rs.b    1       ; a byte to hold the maximum anims
```

The basic idea, then, is that for each game cycle we increment the

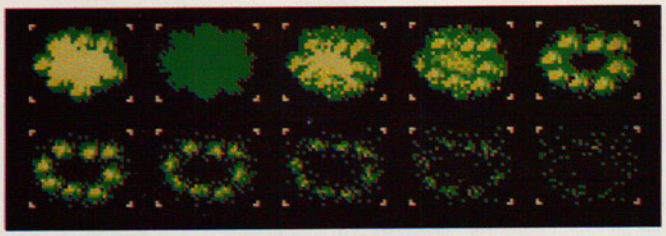


FIGURE ONE:
AN EIGHT-ANIMATION *MENACE* ALIEN

Anim.Num (0,1,2,3,... etc) until it hits the value in Num_Anims when we would reset it back to zero. Straight away a small problem crops up in that we may not want to animate an alien every game cycle (1/25th of a second for *Menace*). In fact very few of the aliens animate every game cycle as this is a little fast, so we will also have to introduce an animation delay into the structure:

```
Anim_Delay          rs.b 1      ; a byte to hold the animation delay
```

The method of incrementing the Anin.Num until it hits a maximum then resetting it to zero is termed a WRAP animation as it wraps around to zero. In some cases we may want the animation numbers to increase to a maximum and then decrease to zero: this is termed an UPDOWN animation. To indicate whether or not to use a wrap or updown type of animation requires us to store a single bit of information. There are many occasions in the controlling of an alien that only requires a simple on/off definition. These tend to be grouped together in the data structure and called a MODE word or byte. This is simply a collection of bits used for general-purpose description of simple on/off controls. This is now added to the structure:

```
Mode            rs.b 1          ; single byte to hold eight flags

UpDownequ       1<<3            ; define the updown flag as bit 3
```

Note the equate of which bit in the control byte reflects the animation type. NEVER use 'magic numbers' in your source such as the statement

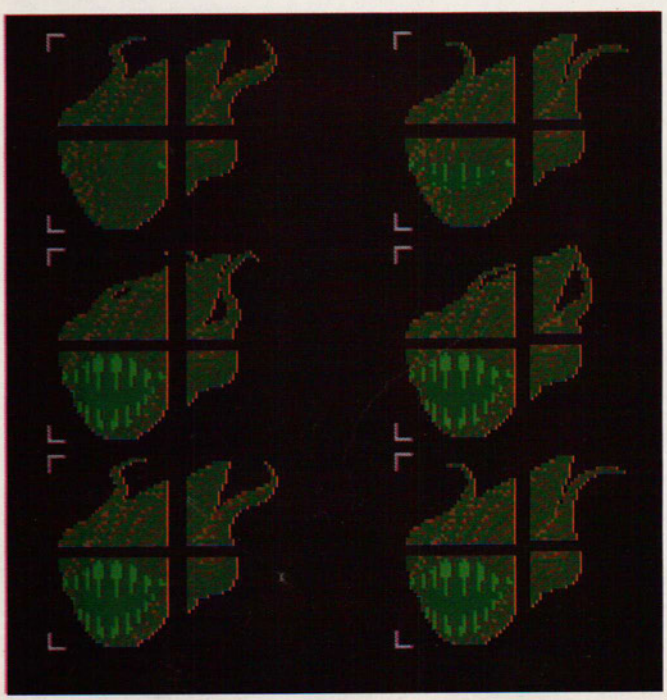
```
    btst    #3,Mode
```

This in no way tells you what bit 3 signifies: you may remember as you are working on the project, but what about in a year's time when you come back to re-use some of this source? The above should have been written as

```
    btst    #UpDown,Mode
```

which is very clear and has the advantage that to change the actual bit number that the UpDown flag is set at requires only the EQU statement above to be changed. Always make the assembler do as much work for you as possible. Right, lecture mode OFF, let's introduce the next element into the structure.

FIGURE TWO:
FOUR SPRITES MAKE ONE LARGE SPRITE



Reason for Living is Dying

Most aliens' sole purpose for existing in a shoot-em-up is to be blasted. Do we want them to die after being hit once, though? For most, I think not. We will have to introduce a variable to define how many times we can fill an alien with laser shot before it explodes:

```
Hits_Num rs.b       1           ; number of shots an alien can take
                                ; $ff for infinite
```

As you can see a certain value has been introduced that makes some aliens indestructible. This is not used very often but is handy, say for some asteroids tumbling across the screen that you simply have to dodge and cannot destroy (anybody who reached Level 6 of *Menace* will know exactly what I mean!)

As far as describing the alien, that is about it. I have made no mention of the size of the alien in the structure as in *Menace* all aliens were 32 x 24 pixels in size. To make larger ones simply meant moving a few about as though they were joined together (see Figure Two). It certainly makes life simpler handling only one size of alien, though you do lose out a little on flexibility.

Back to Motion

Now back to the movement of the aliens. The basic GOTO command was implemented simply by defining pairs of x,y coordinates for the aliens to head for – they continued until they were destroyed or until a special coordinate value told them to stop.

This is very inflexible for performing, say, a small circle movement. We would have to define goto points for many points on the circumference of the circle. This may take quite a while to work out, and would have to be recalculated for any other aliens at different x,y coordinates on the screen that wanted to perform a similiar circle.

One simple way of overcoming this problem is to use a relative coordinate system whereby rather than interpreting the x,y values as goto coordinates, they are simply offsets that are added to the present x,y coordinate. This allows us to define a shape as a set of offsets from a base point, the base point being arbitary. This should be able to be used in conjuction with the goto points so we can switch freely between absolute and relative movement at any time.

It is apparent that some form of control data has to be added into the coordinate list to switch between relative and absolute or one may be mistaken for the other. What basically happens in *Menace* is that movement always starts off in absolute mode, but at any time we can switch to and from relative mode by making one of the coordinate values a control word. For example to goto the middle of the screen, then move in a triangle shape, then return again may look something like:

```
    dc.w  160,100       ; goto 160,100
    dc.w  OffsetMode    ; switch to offset mode
    dc.w  2,2           ; move 2 along and 2 down a few times
    dc.w  2,2
    dc.w  2,2
    dc.w  -2,0          ; move 2 back a few time
    dc.w  -2,0
    dc.w  -2,0
    dc.w  2,-2          ; move 2 along and 2 up a few times
    dc.w  2,-2
    dc.w  2,-2
    dc.w  OffsetMode    ; toggle back to absolute mode
    dc.w  0,100         ; back to the start x,y
```

Control words have certain bits set that distinguish them from a goto coordinate (not much point in, say, having 32768 as a coordinate with a screen width of 352 pixels). What we end up with is a small language that describes the path aliens follow. We can go on adding control features

that make the path data more compact yet more powerful. The offset mode will set a bit in the MODE byte to indicate when it is in offset mode.

Memory Economy

To cut down the amount of data required to describe a complex path (like a large ellipse formed from offsets) it would be ideal to be able to branch to some other path data, and then return after executing so many of its commands. The triangle path above, for instance, requires 40 bytes for the offset data. If we required 12 aliens to perform the triangle movement, but at different coordinates on the screen, we only want to define the offset data once, and have the rest of the aliens branch off to that data inbetween executing their own path data. Menace has a GOSUB command in the path data for this purpose. The gosub command passes over to the new offset data, then at the end of the data is the equivalent of a RETURN command which returns control back to original path data. The above could be written as:

```
        dc.w    160,100             ; goto 160,100
        dc.w    Gosub,Newpath-*     ; branch to some new data
        dc.w    0,100               ; back to the start x,y


Newdata dc.w    OffsetMode  ; switch to offset mode
        dc.w    2,2         ; move 2 along and 2 down a few times
        dc.w    2,2
        dc.w    2,2
        dc.w    -2,0        ; move 2 back a few times
        dc.w    -2,0
        dc.w    -2,0
        dc.w    2,-2        ; move 2 along and 2 up a few times
        dc.w    2,-2
        dc.w    2,-2
        dc.w    OffsetMode  ; toggle back to absolute mode
        dc.w    Return
```

If we had 12 similar paths to the above the memory required would now be drastically reduced.

More Memory Economy

Another useful memory saver is a FOR type contruct that allows any part of the path data to be repeated a set number of times. Imagine trying to move a series of aliens in a 'stepping' motion whereby they move, say, along 16 pixels then down 16 pixels, achieved by:

```
    dc.w  OffsetMode
    dc.w  4,0
    dc.w  4,0
    dc.w  4,0
    dc.w  4,0
    dc.w  0,4
    dc.w  0,4
    dc.w  0,4
    dc.w  0,4
```

The above performs one step, but if we wanted to repeat this 10 times, it would normally mean repeating the data 10 times or using many GOTO statements – but, yes you've guessed it, a loop function was implemented: simply appending the command

```
    dc.w  DoLoop
```

to the end of the data will cause the last xx bytes of path data to be repeated so many times. This caused two new items to be added to the alien structure, namely

```
Loop_Offset     rs.b 1      ; the number of bytes to loop back
Loop_Count      rs.b 1      ; how many loops to perform
```

This limits the loop command to only allowing one size of loop in each individual path, although this restriction was not noticed as multiple loops were never required.

Other Tweaks

A PAUSE function was added that allowed an alien to pause at a specific point for any length of time. A simple function but used quite often:

```
    dc.w  Pause,PauseValue ; the Pause value is in 1/25 seconds
```

A SEEK mode was added that allowed an alien, at any time, to start to track your ship and try to collide with it to reduce your shield. This is worked similarly to the pause function with a count specifying how long to seek your ship before carrying on with the rest of the path data:

```
    dc.w  Seek,SeekCount      ; seek ship for 1/25 * SeekCount secs.
```

Related to the seek command above is the FIRESEEK command which allowed one alien to start another path. This was used to allow aliens to fire missiles that used the seek mode:

```
    dc.w  FireSeek,SeekCount ; start a seek path from this alien
```

To finish off the command set some additional simple commands that came in useful were added. These are:

ChangeSprite – allowed you to change the sprite number at any time. Useful for transforming aliens into other types.

ChangeSpeed – quite obvious this one. Slow moving sprites can be given a quick turbo boost to attack the ship.

ChangeAnim – allows manual changing of the animation number. Used to mimick, say, turning a corner at any point on the screen

And that, basically, is that. Table 1, below, lists the full data structure for an alien path. Hopefully you can see the benefit of designing such a control system for moving objects about. It need not necessarily only be used in a shoot-em-up style game but can relate to many styles of game. It soon becomes quite simple to add more powerful commands to the code, which results in the code being useful in many more projects.

Now you know what the commands do – next month I will present the source and explain how they are accomplished. ■

TABLE 1
FULL DATA STRUCTURE

```
    rsreset
Next.Path       RS.W    1   offset to the next path
X.Pos           RS.W    1   current x position
Y.Pos           RS.W    1   current y position
Kills.What      RS.W    1   kills others aliens if dead (0-11)
Table.Offset    RS.W    1   the current table offset
Sprite.Num      RS.B    1   sprite number
Anim.Num        RS.B    1   animation number
Anim.Delay      RS.B    1   delay in 1/25th secs, dynamic copy
Anim.Delay2     RS.B    1   static copy to refresh the above
Speed           RS.B    1   speed in pixels
Pause.Count     RS.B    1   dynamic pause counter
Mode            RS.B    1   flags, see bleow
Loop.Offset     RS.B    1   loop offset in bytes (-ve)
Loop.Count      RS.B    1   dynamic loop count
Hits.Num        RS.B    1   number of hits to kill
Num.Anims       RS.B    1   number of animations
Seek.Count      RS.B    1   dynamic seek count
Table.Size      RS.B    0
```

* This is followed by x,y bytes to move to (always even) with command

* codes inserted to alter the control and movement.

```
*   Mode bits
    Offset.Mode     equ     1<<0    set when in offset mode
    Seek.Mode       equ     1<<1    set when in seek mode
    UpDown          equ     1<<3    set when updown
                                    animation
    AnimUpDown      equ     1<<4    set to animate up,
                                    reset down
    HeatSeekPath    equ     1<<5    set to signify a
                                    heatseeker
```