# AMIGA
## FORMAT

**BIGGEST EVER!**

## tape...

**NO TAPE?
DEMAND ONE FROM YOUR NEWSAGENT NOW!**

## disk...

# music

**SMASH the sound barrier
with our complete guide
to starting out in music**
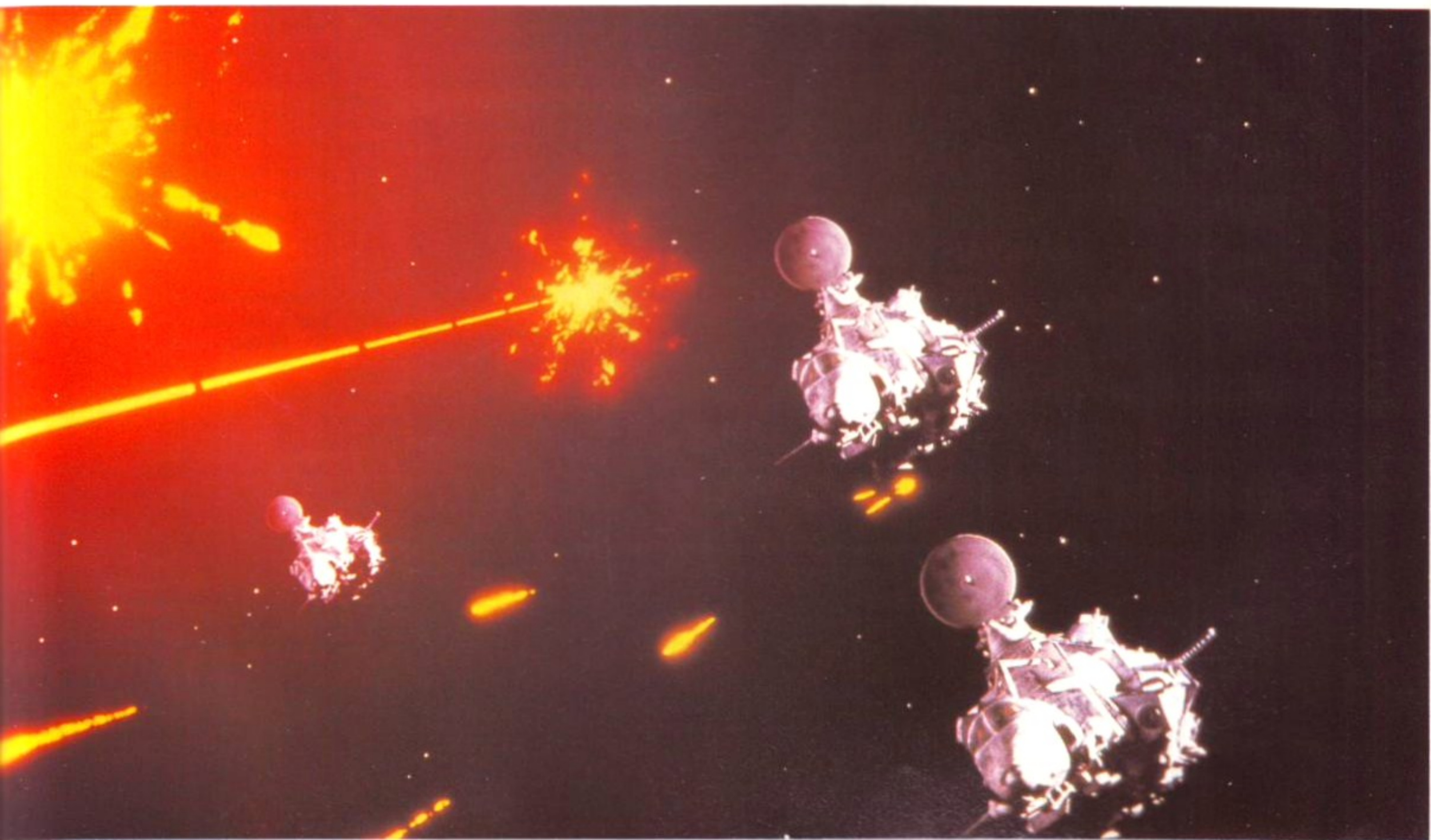
## football

**Get a KICK out of our
World Cup special**

**NO COVERDISK?
DEMAND ONE FROM YOUR NEWSAGENT NOW!**

07

9 770957 486004

# THE WHOLE TRUTH ABOUT
# games programming
## PART 5
# Collision detection

The *Menace* code is nearly complete, becoming playable as **DAVE JONES** adds the bump-into-things routines.

This month sees all of the game coming together – and now being playable! – with the inclusion of the collision detection routines.

There is normally some form of collision detection performed in all games – one object moving into another object is a very common occurence that normally has to be detected. *Menace* required the following collisions to be identified:

- Player ship to aliens
- Player ship to foreground (expert mode and guardian)
- Weapons to aliens

These three are really all that is required in *Menace*. The first two of the above are very simple thanks to the Amiga hardware and the way *Menace* was designed.

Remember that the player's ship is a hardware sprite. The Amiga has the ability to detect collisions between any of the harwdare sprites and any bit plane(s) we specify. In the dual playfield mode that *Menace* runs in, bit planes 1, 3, & 5 form the back playfield, with the even planes (2,4 & 6) forming the front playfield. The aliens are drawn into the back playfield (planes 1,3 & 5) so we want the hardware to detect when the hardware sprites that form the ship collide with pixels in these planes.

A problem springs up here, though, in that the actual background graphics would register a collision as these completely fill up planes 1 and 3 – being four-colour, they only require two planes. Plane 5 is, however, untouched by the background graphics and will ONLY contain data from the actual aliens. So rather than detect a hardware sprite collision between planes 1, 3 & 5 we will only detect a collision to plane 5.

You may notice a small quirk here in that only checking the third plane in the alien graphic data will not be very effective if the alien used very few colours that have bits in the third plane set. In reality all of the aliens in *Menace* used mainly the colours that are included in the third plane set (colours 4, 5, 6, & 7) as these were redefinable for each alien. The background colours (0,1,2 & 3) were very rarely used in the aliens themselves.

### Hardware Registering
The hardware register that controls the sprite-to-playfield detection is called CLXCON. This lets you select which bit planes to detect, and the value to detect (normally 1 for a set pixel). All that happens is that a mask of the ship is ANDed with the bit plane data. This will only register a collision when two bits overlap, which is indicated in the CLXDAT register.

This is simply a hardware implementation of a commonly-used software routine where an object's mask

is ANDed with a screen mask to check for a collision. This type of detection is termed 'pixel perfect' detection as just one pixel of your ship hitting a single pixel of an alien will be detected. Many games use other methods of detection that are sometimes a little crude. How many times have you heard the shout 'No way I was dead, I was miles away from that bullet, stupid ❖◆❋❊■❋ game!'.

### Ship to Foreground
The ship to foreground collision detection was handled simiarly, with a collision between plane 6 and the ship being registered. Initially I registered collisions between planes 2, 4 & 6, all of the foreground planes. This then exhibited a quirk because the ship missiles are drawn into the foreground playfield. If the ship had quite a few speed-ups you could fly over some of your own slower-moving missiles – and then you lost energy!

Obviously not a desired effect, so the solution was to only draw the missiles into planes 2 and 4, and collision detect only to plane 6. This gave quite a lenient feel to the foreground collision detection as colours 0,1,2 & 3 in the foreground graphics would not register a hit. There was many a sprinkling of the other colours in the graphics, though, so the effect was not too noticeable.

As a hint, if you want to see the sort of effect that using only a single plane of some graphics would have in collision detection, load your graphics into *Pixmate* or *Butcher* (image processing programs) and switch off the bitplanes you will not be using. This gives you a visual indication of just what is going to register a hit. It also gives you the opportunity to swap a few of the colours around to get the best use from the single plane.

### Being Generous
It is usually preferable to make collision detection 'generous'. This simply means that the player can stray into objects by a few pixels before a collision is detected. This is not possible to do if you are using the hardware collision detection, but is quite simple if you are implementing it in software.

Figure 1 shows the main sprite from Level One of *Blood Money*. Beside it is its collision mask. Rather than use the actual mask of the sprite, a new mask was stored that had a few pixels trimmed off around the edges. The blitter, rather than the hardware method, was used to AND this mask with a plane of the screen.

This allowed certain parts of your helicopter (note in the diagram the blades have been removed) to enter the side walls with safety. This allows a little more skill into the game and eggs players on to have a go at squeezing through tight spots.

This method of collision detection has one major downfall. Although we may detect we have hit an alien, we do not know which one. For *Menace* this does not matter as the game was kept nice and simple, with collisions with any alien simply reducing your energy – the only exception being the bonus icon that you shoot and pick up. The bonus icon, though, can only appear on its own on the screen, so if we hit an alien during the bonus routine, it must of course have been the bonus icon.

### What hit What?
The last part of the collision detection, missiles to aliens, must be able to determine what particular missile hit what particular alien. This must be implemented using some



Figure 1
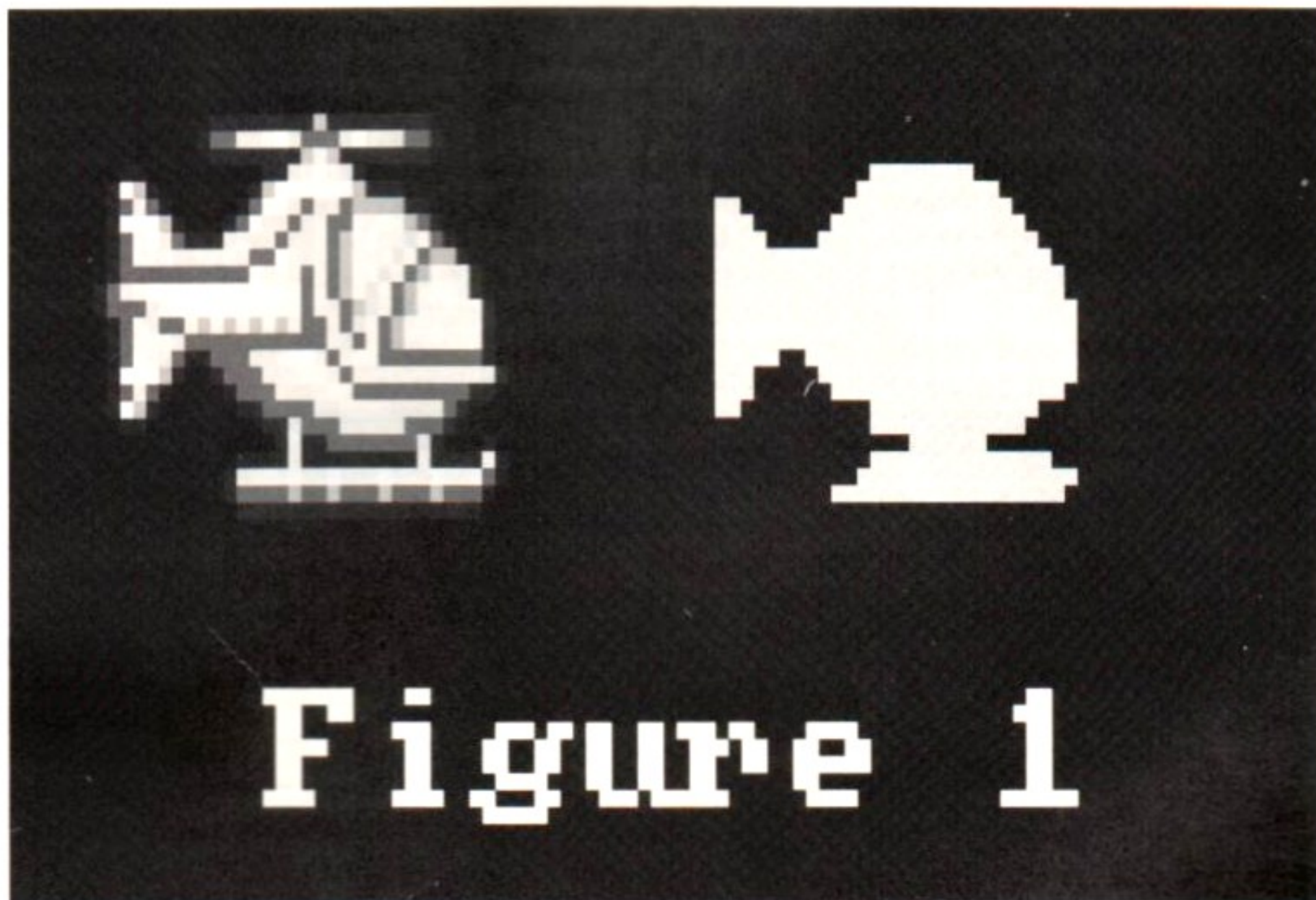
form of coordinate checking. This is the second main way to perform collision detection.

Coordinate checking can never be pixel perfect unless we checked every pixel coordinate in the missile to every pixel in an alien, which would require HUGE amounts of time and cause a game to crawl along. What normally happens is that a 'box' is defined within the two objects you are checking. A box only needs two coordinates, the top left and bottom right. If two boxes overlap then we flag a collision. You can see this is quite rough in that not many aliens appear as a box shape.

In *Menace* the missiles are quite small, so rather than define a box for each missile, only one coordinate at the tip of each missile was checked. If this coordinate entered the box of any alien then a hit was registered.

I made all the alien collision boxes 32 x 24 pixels in size, exactly the same as the alien size. This would appear to cause problems if the alien was quite small in size, not filling the entire box. Have a look at Figure 2. This shows a small alien with a missile entering its box, flagging a collision. But as you can see we would not want this particular case to flag any collision as the missile is quite far away for the actual alien itself.

One way around this would be to store a collision box size for each alien. The one in Figure 2 for example would suit a box of 24 x 18 better. This makes the calculation of whether a missile was in a box or not more lengthy, though, as we are not dealing with constant numbers. This may seem being picky but if we have 12 aliens flying around, and as you can have up to 18 different missiles active at once (missiles, laser, canons etc) this would mean checking every missile against every alien = 216 checks. This will be quite time consuming as there is a fair bit of calculation in each check.

### Make it Simpler

What we need is a quick way to tell if a missile has hit ANY alien; if this happens we can then check for WHICH alien it has hit. Fear not, for this is quite simple to do.

What we need to do is combine the mask detection with the coordinate detection. We cannot use the Amiga's hardware detection for missiles to aliens as this only works with hardware sprites. The blitter can do a quick check for us, though.

The blitter has a bit called BZERO in the DMACONR register. This bit will be set true if the result of the previous blit operation is all zeroes. We therefore get the blitter to perform an AND operation between the mask of the missile, and plane 5, which only contains data from the aliens. If the BZERO bit comes back false, ie the
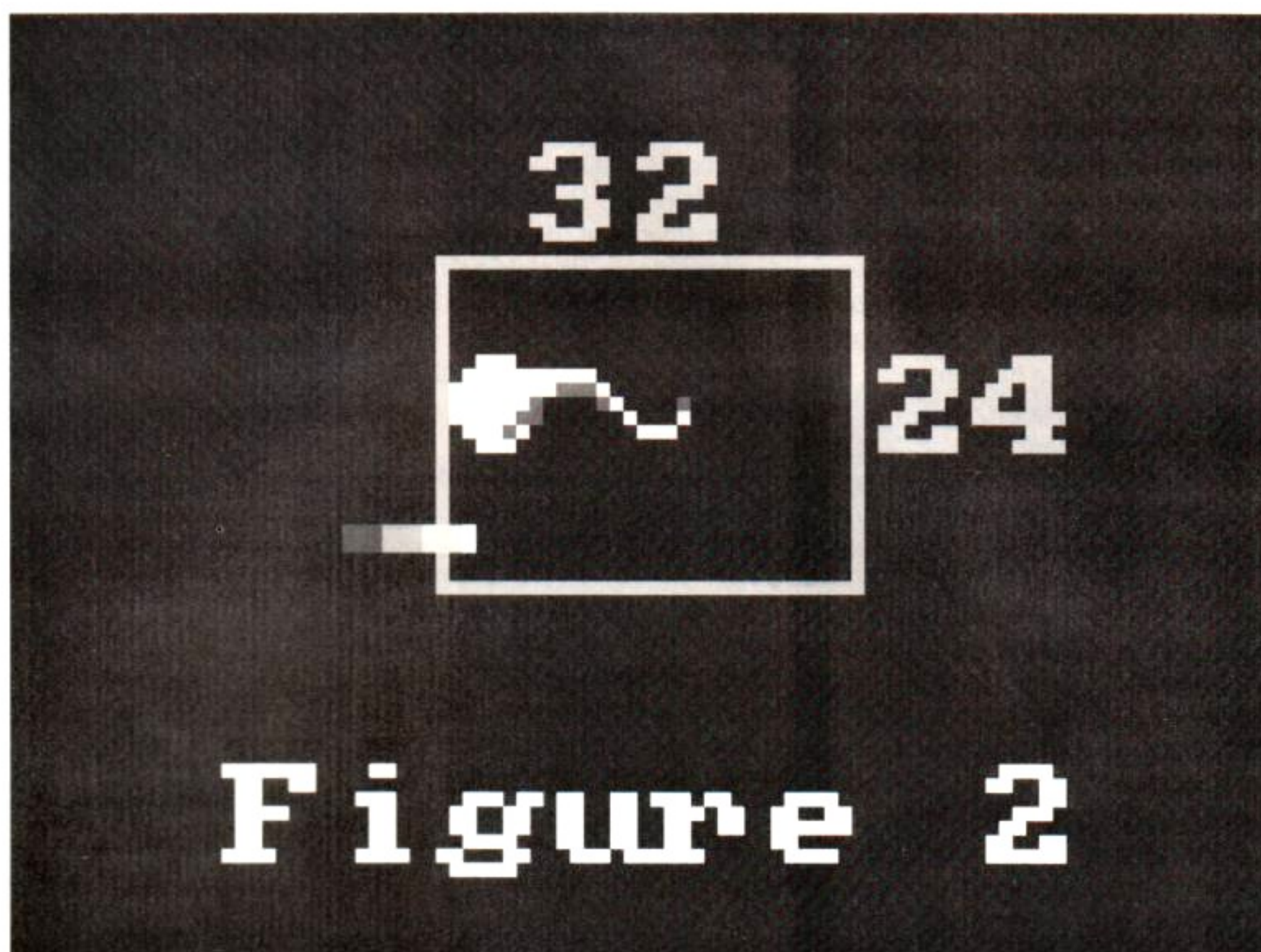


result of the blit was NOT zero, then the missile must have collided with some alien data.

NOW we can perform the coordinate detection to find out which alien we hit. This also solves the problem in Figure 2, which would not now flag a collision because the mask detection would not flag a hit, so therefore the coordinate detection would not be then be performed.

When we perform the blit to AND the mask with plane 5 we switch off the blitter Destination channel. We do not want the blitter to draw the result of the blit, so it is pointless to have an area of memory to point the D channel to.

It is quicker and therefore more efficient to switch off the Destination channel for this blit. This is a handy feature of the blitter allowing us to combine a few channels in some way, but only being interested in whether the result was zero or not.

This method of performing the missile-to-alien collision will be considerably faster, even taking into account the blitter operation. On average only three out of the eighteen missiles would actually be hitting aliens. This reduces the coordinate checking to a lower level: only 3 x 12 = 36 comparisons.

### Love Missile F-111

That, then, is all the collision required in *Menace*: ship to aliens, ship to foregrounds, and missiles to aliens. The source code for this month (on next month's disk) covers all this, along with the drawing of the missiles. There are four types of missile:

- Normal missile
- Cannon

**Figure 2: An alien from *Menace* is struck by a missile. If a standard-sized box is used to define the aliens' collision area, this rather small alien will be hit too easily. A smaller size of box is needed for smaller aliens.**

- Laser
- Outrider

All have different ranges and speeds. They all simply subtract one from an alien's hit number, but lasers are not killed once they have hit an alien – they carry on till the end of the screen – and are therefore more powerful. All other weapons are killed as soon as they hit an alien.

Graphically they are quite small (typically 16 x 4 pixels) and are just four-colour. The blitter is used to draw them into plane 2 and 4. Weapons in shoot-em-ups nowadays are getting BIG. If you plan on trying to write one, I would say the bigger the weapons the better for MEGA effects. There should be no difference in the collision detection, only in the extra time taken to draw bigger, badder weapons.

### Completing the Code...!

Sadly, there was no space for the code on this month's Coverdisk, but watch out for it next month – the *Menace* code is now pretty well complete and playable!

All the paths for Level One are there to be blasted, collisions to aliens will reduce your ship energy until zero, at which point you will be returned back to the CLI. When you reach the end of the level the game will also return to the CLI.

*This leaves only the guardian to be added, which will be covered next month along with a discussion of the ancillary stuff such as music (where does it come from, how much memory should you reserve, how much does it cost! etc), text routines, disk routines and such like.* ■