JANUARY 1993 AMIGA FORMAT 5

Your Turn! Part 4

The Bullfrogs' coding methods aim to make life as easy as possible. And if you've been following this series you'll know just how easy it is to create your own game by following their advice.

How to program your own games in assembler

Welcome to the fourth part of our series teaching you how to program top-class games using the professional techniques of the Bullfrog programming team. And for this month's tutorial **Scott Johnston** shows you how to write collision routines for the baddies – no more walking through trouble folks. He also offers a spot of expert advice on how to design and add some extra levels to your game...

WELCOME BACK. Run the demo and you will notice several things. There is a horrible *SoundTracker* routine playing in the background – thanks to Kevin for that.

You can also now walk around the four levels, and on completion of all four you will go back to the start. (You can add extra levels in yourself, if you're daring. A little more advice on that later.)

You also have three headless *Populous 2* men wandering around the landscape. You can run right through them at present, so you need not worry too much about them when you're playing the game.

On to the task for this month. You are now going to write the collision routine for the baddies, the headless *Populous* peeps. If you like, you can try writing it before you look at the routine below.

You have to keep in mind that the hero's x and y details are stored as a scaled value four times the size of the true screen x, y. You must also keep in mind the fact that your man has a set width, as have the bad guys. In fact you can use the following equates to control the collision size:

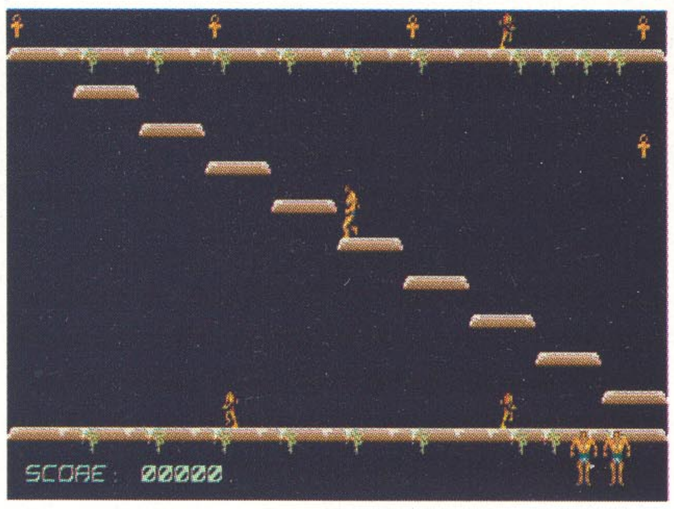
```
MAN_WIDTH       How wide is your man
MAN_HEIGHT      How tall is your man
BAD_LEFT_WIDTH  Where the left side of the bad guy starts.
```

Continued overleaf

```
BAD_RIGHT_WIDTH          Where the
right hand side of the bad guy is
BAD_TOP_HEIGHT           Where the
top of our bad guy starts
BAD_BOTTOM_HEIGHT        The bot-
tom of the bad guy is found here.
```

The general routine is based on the Ankh collision routine and is shown below. Sorry that it is so long. You can leave out the comments if you want, but when you come back to it in the future it will be harder to understand. This routine should be placed in file move.s between the _bad_collision label and the rts instruction. There is a cheat on level four. When you fall through the hole in the floor, instead of vanishing off the screen like you used to, you will now reappear at the top of the screen. Be careful, because you can still fall too far and die. See if you can change the code so that instead of reappearing, you die, lose a life and the level is re-initiated.

Here's what you've got so far. Notice the baddies now appear at the bottom of the screen. They may look familiar, which is no surprise because they're straight out of *Populous 2*.



We need an end sequence because when you die the third time, you have to quit and reboot the computer in order to replay the game. If you can work this out, do so, and also try this: see if you can add a high-score table to the game. If anyone is interested, I got 6750 before I lost my first life.

Don't worry too much if you can't get the high-score table sussed. There will be one in next month's issue, along with a small options screen at the start. Meanwhile, you can change any of the code in this month's issue. Try, for example, to have the bad guys moving up and down on the screen, instead of left and right.

COLLISION ROUTINE CODE

```
tst.w   died    ;are we in the middle of dying?
bne.s   .finished       ;yes then we can't col-
lide with anything
cmp.w  #NO_LIVES,game_over      ;is   the   game
over?
bgt.s   .finished       ;yes then we can't col-
lide with anything
lea  _bad_guys,a0       ;point at the bad guys'
structures
move.w #MAX_BADDIES-1,d0 ;number  of  baddies  to
look at
looptst.w BAD_ON(a0)    ;is this guy turned on?
beq.s   .next   ;no then move to the next one
movem.w BAD_XY(a0),d1/d2 ;pick up x and y position
of man
movem.w man_x,d5/d6     ;pick up the hero's x and
y position
asr.w   #FOUR,d5 ;scale down the x
asr.w   #FOUR,d6 ;scale down the y
move.w d1,d3    ;take a copy of bad x
move.w d2,d4    ;take a copy of bad y
add.w   #BAD_RIGHT_WIDTH,d3     ;find  the  right
hand side of the bad guy
cmp.w   d3,d5   ;and see if we are on the left
hand edge
bge.s   .no_collision   ;no then we can't collide
add.w   #BAD_BOTTOM_HEIGHT,d4   ;find the bottom
side of the bad guy
cmp.w   d4,d6   ;are we above this, ie smaller
number?
bge.s   .no_collision   ;no then we can't collide
add.w   #MAN_WIDTH,d5   ;increase man's x posi-
tion by his width
add.w   #BAD_LEFT_WIDTH,d1      ;and   find   the
left side of the bad guy
cmp.w   d1,d5   ;are we to the right of this?
ble.s   .no_collision   ;no then we can't collide
with the man
add.w   #MAN_HEIGHT,d6  ;find the height of hero
add.w   #BAD_TOP_HEIGHT,d2      ;and  the  height
of the bad guy
cmp.w   d2,d6   ;are we below the top of our bad
guy?
ble.s   .no_collision   ;no then we can't collide
again
;if we have got to here, then we have collided
move.w  #0,man_vx       ;so   cancel   the   mans
velocity
move.w  #1,died ;set the died flag
move.w  #MAN_CRUMBLE_START,man_frame    ;and  the
animation frame
.no_collision.nextlea  BAD_SIZE(a0),a0  ;move  on
to the next bad guy
dbra    d0,.loop;until there are no more to check
.finished
```

TO ADD EXTRA LEVELS

Design your new level inside of data_c.s, making sure that there are 13 lines, each with 20 bytes on them. At present there are only five types of block in the game and these are:

1. Left edge of platform
2. Middle section of platform
3. Middle section of platform
4. Right section of platform
5. Stand-alone platform

After the level is created you need to tell the program it exists. If you increase the equate MAX_LEVELS and add a new jump label to the jump table in init.s then create a new section label and rts command to the end of the file, then you should be able to jump around on the new level. At the moment there are no collectables and no bad guys. To place collectables you need:

```
lea     _objects,a0
move.w  #1,OBJ_ON(a0)
move.w  X,OBJ_X(a0)
move.w  Y,OBJ_Y(a0)
```

for the first one, where X is the x position, and Y is the y position of the object. To place more after this use:

```
lea     OBJ_SIZE(a0),a0
move.w  #1,OBJ_ON(a0)
move.w  #X,OBJ_X(a0)
move.w  #Y,OBJ_Y(a0)
```

Finally, you need to tell the computer how many collectables there are on this level with a line like this:

```
.move.w #NUMBER,to_collect
```

Where NUMBER is the number of objects that you need to collect to finish a level. Remember, this need not be as high as the actual number of objects on the level. Take the first level, for example. There are five objects on the screen, but you only need to collect four of them.

Placing baddies is almost the same. To start with you do this:

```
lea     _bad_guys,a0
move.w
#BAD_STATE_LEFT,BAD_ON(a0)
move.w  #X,BAD_X(a0)
move.w  #Y,BAD_Y(a0)
move.w
#BAD_LEFT_START,BAD_FRAME(a0)
```

Slightly more complex. When placed the man must be placed on to a platform, otherwise he wobbles in mid air. BAD_ON(a0) is loaded with the current state of the man and can be a choice of the following:

```
BAD_STATE_LEFT  man   is   walking
left
BAD_STATE_FROM_LEFT     man     is
facing  forwards,  and  came  from
the left
BAD_STATE_FROM_RIGH     man     not
moving came from right
BAD_STATE_RIGHT the man is walk-
ing right.
```

Similarly, BAD_FRAME(a0) must be loaded with the current animation, and you have these to choose from:

```
BAD_LEFT_START  start  of  anima-
tion going left
BAD_LEFT_END    end of animation
going left
BAD_MIDDLE      bad guy faces the
screen
BAD_RIGHT_START start  of  anima-
tion for right movement
BAD_RIGHT_END   end of animation
for going right
```

That's the first man. To place more than one man down you must move into the array and insert your next man using something like this:

```
lea     BAD_SIZE(a0),a0
move.w
#BAD_STATE_RIGHT,BAD_ON(a0)
move.w  #X,BAD_X(a0)
move.w  #Y,BAD_Y(a0)
move.w
#BAD_RIGHT_START,BAD_FRAME(a0)
```

When you have all your men down you don't need to declare the number, but you do have to make sure there is a return from subroutine at the end of your code. Then compile and away you go.