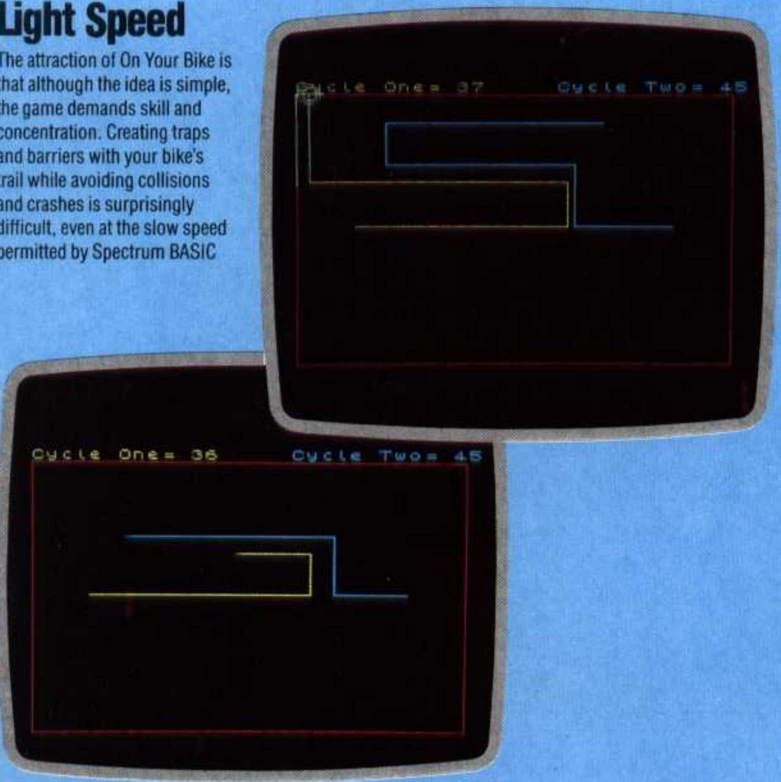
ON YOUR BIKE

Though most successful commercial games are fairly lengthy, and written in machine code for speed, it is possible to write an entertaining game in BASIC. The game that follows is fairly simple, and indeed takes up only 35 lines of BASIC, but is still good fun to play. Furthermore, it is a two-player game, so is less anti-social than most!

The game is called 'On Your Bike' and is based on a scene from the classic Walt Disney film *Tron.* It is a contest between two opponents that requires skill and fast reactions and takes place in an enclosed arena. You each have a bicycle that travels at an incredible speed and cannot be stopped. Your only control allows you to turn through 90 degrees at top speed.These bicycles leave solid walls of light in their trail, and the object of the game is to force your opponent to crash in the ever-tightening maze you create as you zoom around the arena.



Light Speed
The attraction of On Your Bike is that although the idea is simple, the game demands skill and concentration. Creating traps and barriers with your bike's trail while avoiding collisions and crashes is surprisingly difficult, even at the slow speed permitted by Spectrum BASIC

IAN McKINNELL

The game has been implemented on the ZX Spectrum, which is not known for the speed of its BASIC. As this is an action game, the program has been designed for speed rather than elegance,so much of the listing may seem a little unstructured. Subroutine calls, and other structured devices have been avoided as they would have sacrificed execution speed.

The first stage is the design of the arena and score display. As you can see this is fairly simple, which contributes to the shortness of the final program.The only point to note is that the border of the arena is now one character in from the usable screen area. This is to ensure that the graphics resulting from a collision with the arena wall do not go off screen:

```
10 LET p=0: LET q=0
100 BORDER 0: PAPER 0: CLS
110 PRINT AT 0,1; INK 6; "Bike One= "; q
120 PRINT AT 0,19; INK 5; "Bike Two= "; p
130 INK 2
140 PLOT 8,8: DRAW 239,0: DRAW 0,159
150 DRAW -239,0: DRAW 0,-159
```

The arena has been drawn in red, and we have chosen yellow to represent bike one, and cyan (blue) for bike two. The variables p and q hold the current score for the two contestants.

The next stage is to initialise all variables, and here we have to start thinking about how we are to implement the main action of the game. The action for a single bike is fairly straightforward, and is shown in the flowchart. Using POINT we check if the bike's current position is occupied, and move to the collision routine if it is. If it is not, we move into that position using PLOT, and then read the keyboard to check for any change in direction. Our position is then incremented by one in our current direction, and the cycle begins again. We therefore need four variables: two for our current x and y co-ordinates, and two for our current direction along the x and y axes.

However, we are dealing with two bikes moving at the same time. An elegant solution would be to use four two-element arrays, x(2) and y(2), for the positions for example, but this would slow the game down so we have to use eight separate variables:

```
200 LET x=40: LET y=88
210 LET m=215: LET n=88
220 LET a=1: LET b=0
230 LET i= -1: LET j=0
```

This sets the initial positions of the bikes, and sets them moving towards each other one pixel at a time. The basic action of the game is then fairly simple to implement:

```
400 IF POINT (x,y)=1 THEN LET col=6: GOTO 700
410 IF POINT (m,n)=1 THEN LET col=5: LET x=m:
   LET y=n: GOTO 700
420 PLOT INK 6;x,y:PLOT INK5;m,n
```