velocity and so on. These can be very simple or very complicated, depending on how detailed and accurate we wish to be. For the purpose of this game, we've kept them fairly simple.

The main thing we need to know is the height of the spacecraft. Obviously, it is continually moving, either dropping due to gravity or accelerating away from the planet because of overuse of the rocket engine. To allow us to work out where it is at any one moment, we divide 'time' up into a series of steps or periods.

In each period, we can calculate how far the craft has moved, what the change is in its speed and mass and so on. These periods can be any length you like — the shorter they are, the more accurate the simulation. Once we introduce the idea of periods, writing the equations is easy.

Speed is measured as so many units per hour. In two hours, a car travelling at 10 km/h (10 kilometres per hour) will move 20 kilometres. In three hours, it will move 30 kilometres, and so on. This gives the formula:

Distance = Time × Speed

So in each period we can calculate how far up or down the lander moves by multiplying its velocity by the length of the period (which we define as unity). We can then adjust the velocity by accelerating the craft by the planet's gravitational pull and decelerating it by any push from the rocket motors.

Acceleration due to gravity is always constant (the variable g in the program) and will depend on the planet that is being approached. The illustration shows the values for the planets in our solar system but you could experiment with other values or have the program generate g randomly to provide a more difficult game.

Simulating the rocket motor is slightly more complicated. In this version, the player can burn from one to nine units of fuel in a given period, and the program calculates the resulting acceleration, taking into account the mass of the craft. The exact formula depends on the power of the rocket engines and the type of fuel used. In this program, the figures are chosen to make the game hard to beat; you could try altering them to see how they affect play.

One twist that can be added to the simulation is that it should be played in *real time*. This is a much abused phrase and nowadays usually means that the program (game, simulation, etc.) runs continuously, and never stops for the user to enter data or commands. This is often only the difference between using an INPUT command to gather information and an INKEY$ or GET.

Obviously, Lunar Lander is a better game if the program doesn't come to a halt to calculate how much rocket fuel to burn. If it did this, the player would have time to consider the situation, make a few calculations, and so on. In real life, it would be more a question of quick thinking.

In our lander program, we have a loop that operates once for each time interval in the program. By adjusting the period so that it is actually the time taken to execute the loop, the simulation operates in real time. It takes as long to land the spacecraft in the simulation as it would to land it in real life. Although this is desirable in a simulation, it can be quite hard to get right in a game like this. Usually, the simulation takes too long to be interesting as a game.

There are many things you can do to the basic lander program. The most obvious is to add a graphic display of the descent. Ideas for this vary from a simple round altimeter dial to a side view of a little ship, or even a scaled view downwards of the landing pad. You could also try adding sideways motion so that the craft has to be positioned over the pad as well as lowered onto it.

The craft won't normally drift sideways in space because there is nothing to push or pull it in any other direction than down. But if you're landing on a planet with an atmosphere, you could add in the problem of a surface wind and so on. Some sophisticated versions of the program have several landing areas, situated at the bottom of tunnels and craters so that landing requires plenty of careful steering.

Lunar Lander may seem old hat compared to the fast and furious arcade games of today. But programming and playing it is many people's first encounter with a computer simulation and the whole complex field of making computer programs reflect objects and events in the real world. Programming a moon landing can be the first step to a whole new range of programming projects.

## Touchdown

```
 10  REM Lunar Lander Game
 20  LET g=-1.6
 30  LET t=1
 40  LET f=1000
 50  LET v=0
 60  LET h=2000
 70  LET m=2000+f
 80  LET g=g*t
100  REM Update screen
110  PRINT AT 0,0
120  PRINT "            Lunar Lander"
130  PRINT : PRINT "Height....";INT h;"   "
140  PRINT : PRINT "Speed.....";INT v;"   "
150  PRINT : PRINT "Fuel......";f;"   "
160  PRINT
165  IF h<0 THEN  GO TO 400
170  IF f<=0  THEN  LET f=0: PRINT "*** OUT
OF FUEL       ": GO TO 190
180  PRINT "Key rocket burn 0-9 "
190  LET b=0: IF f>0 THEN  LET a$=INKEY$: IF
a$<>"" THEN  LET b=VAL a$
200  IF b>f THEN  LET b=0
210  LET h=h+v*t
220  LET v=v+g
230  LET v=v+(b*3000)/m
240  LET f=f-b: LET m=m-b
250  FOR i=1 TO 50: NEXT i
300  GO TO 110
400  REM On planet surface
410  IF v>-10 THEN  PRINT "*** Safe Landing
... Well done": GO TO 500
420  IF v>-20 THEN  PRINT "*** CRUNCH! ... y
ou wrecked the lander but the crew survived!"
: GO TO 500
430  PRINT "*** SMASH!  ... Lander destroyed
... no survivors"
440  PRINT : PRINT "You've just blasted a ne
w       crater ";INT (-v*2.1);" Km wide"
500  PRINT : PRINT "Play again (Y/N) ? ";
510  LET a$=INKEY$: IF a$="" THEN  GO TO 510
520  IF a$="y"  OR a$="Y" THEN  RUN
530  IF a$<>"n" AND a$<>"N"  THEN  GO TO 510
540  CLS : STOP
```

**Basic Flavours**

This is a Spectrum listing: other machines do not require use of the word LET. On the BBC Micro, replace line 110 by:

```
10 PRINT TAB(0,0)
```

Replace INKEY$ in lines 190 and 510 by INKEY$(0). Replace VAL a$ in line 190 by VAL(a$). Replace INT h and INT v in lines 130 and 140 by INT(h) and INT(v).

On the Commodore 64 and Vic-20, replace line 110 by:

```
110 PRINT CHR$(19)
```

Replace LET a$=INKEY$ in lines 190 and 510 by GET a$. Replace VAL a$ in line 190 by VAL(a$). Replace INT h and INT v in lines 130 and 140 by INT(h) and INT(v).

On the Oric/Atmos replace line 110 by:

```
110 PRINT @ 0,0
```

Replace INKEY$ in line 190 and 510 by KEY$. Replace VAL a$ in line 190 by VAL(a$). Replace INT h and INT v in lines 130 and 140 by INT(h) and INT(v)