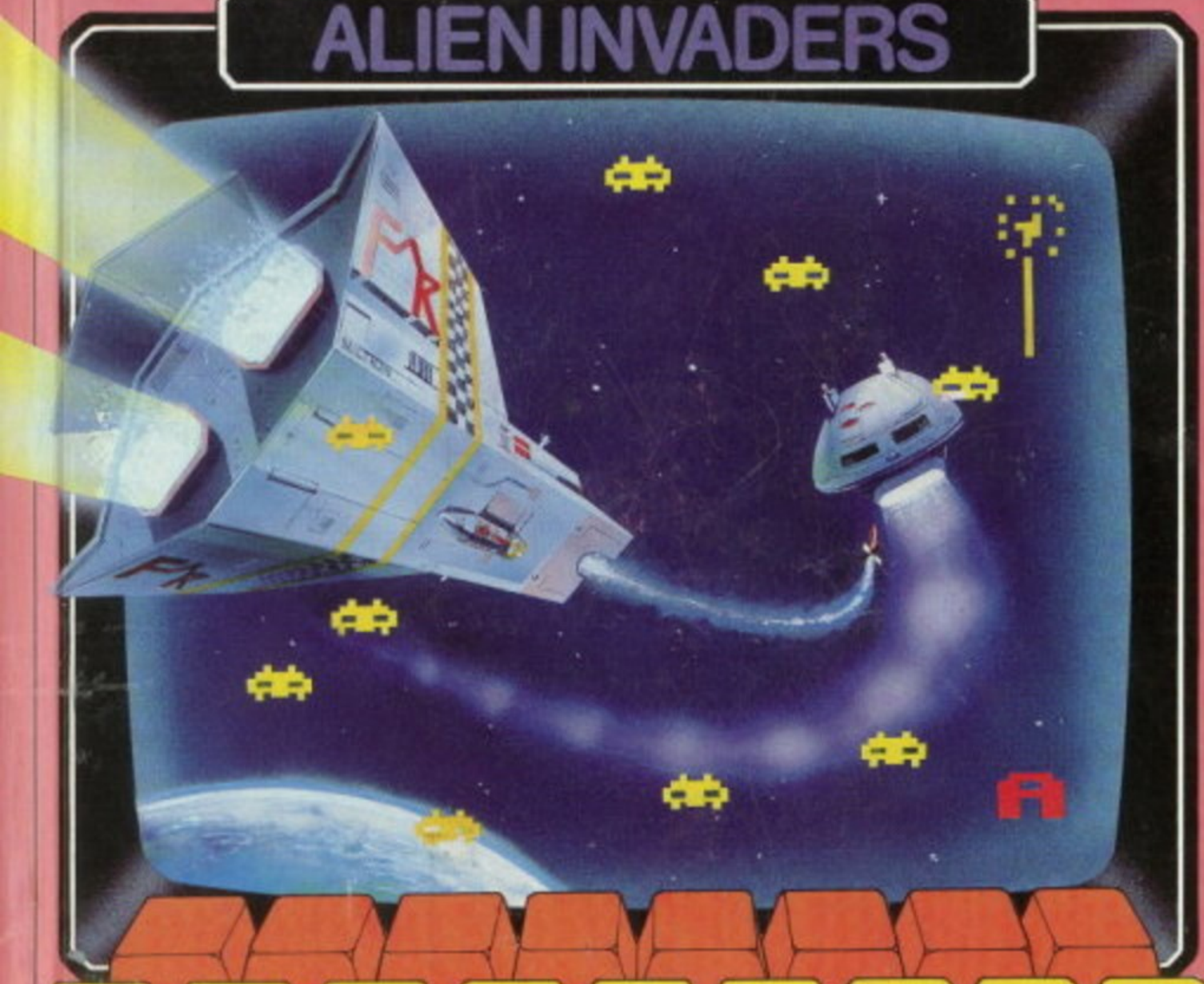
WRITE YOUR OWN PROGRAM

MOVING GRAPHICS

ALIEN INVADERS



FOR COMMODORE 64 AND APPLE IIe COMPUTERS

First published in
Great Britain in 1985 by
Franklin Watts
12a Golden Square
London W1

First published in the
United States in 1985 by
Gloucester Press

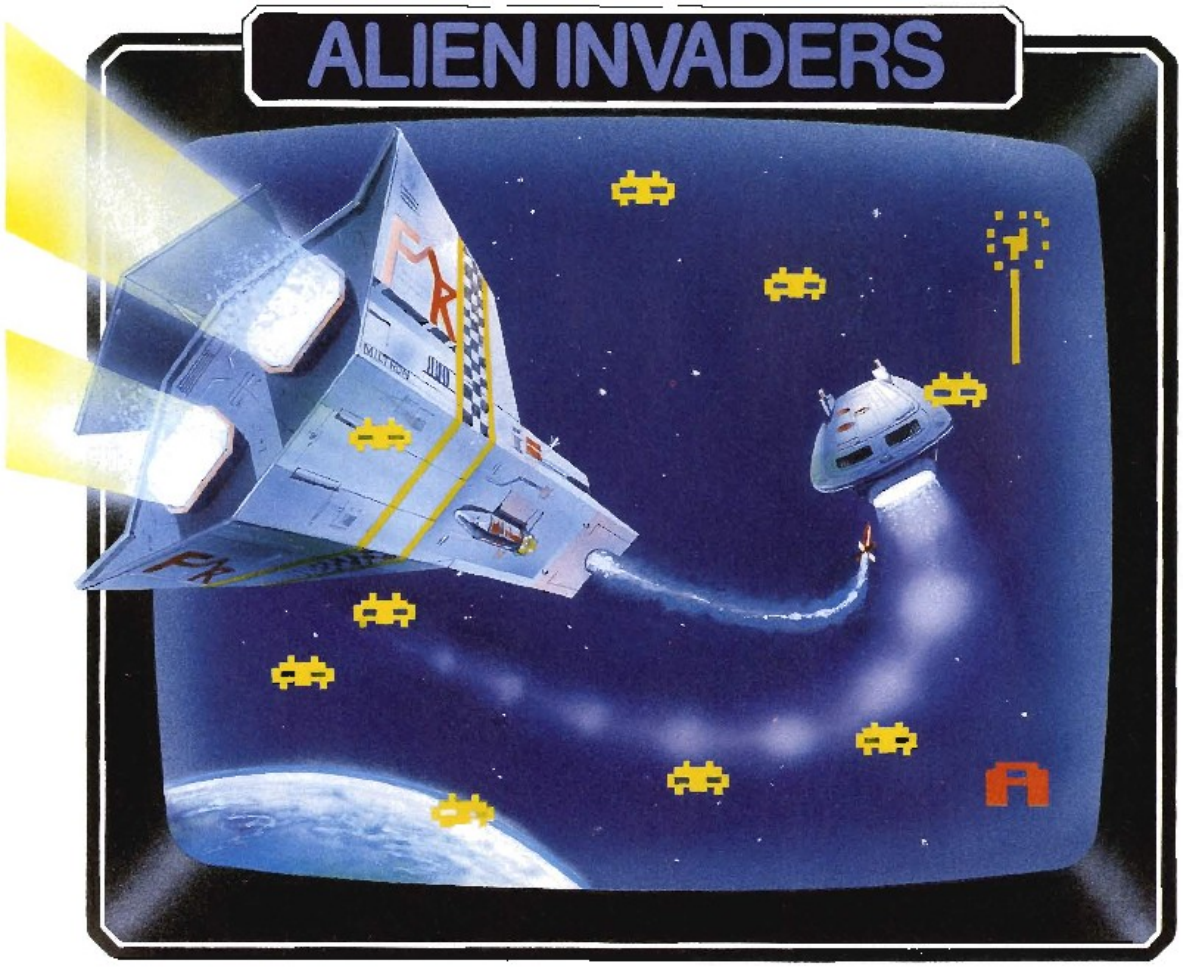
Printed in Belgium

ISBN 0 531 03491 7

Library of Congress
Catalog Card Number: 84-73147

WRITE YOUR OWN PROGRAM

MOVING GRAPHICS



Marcus Milton

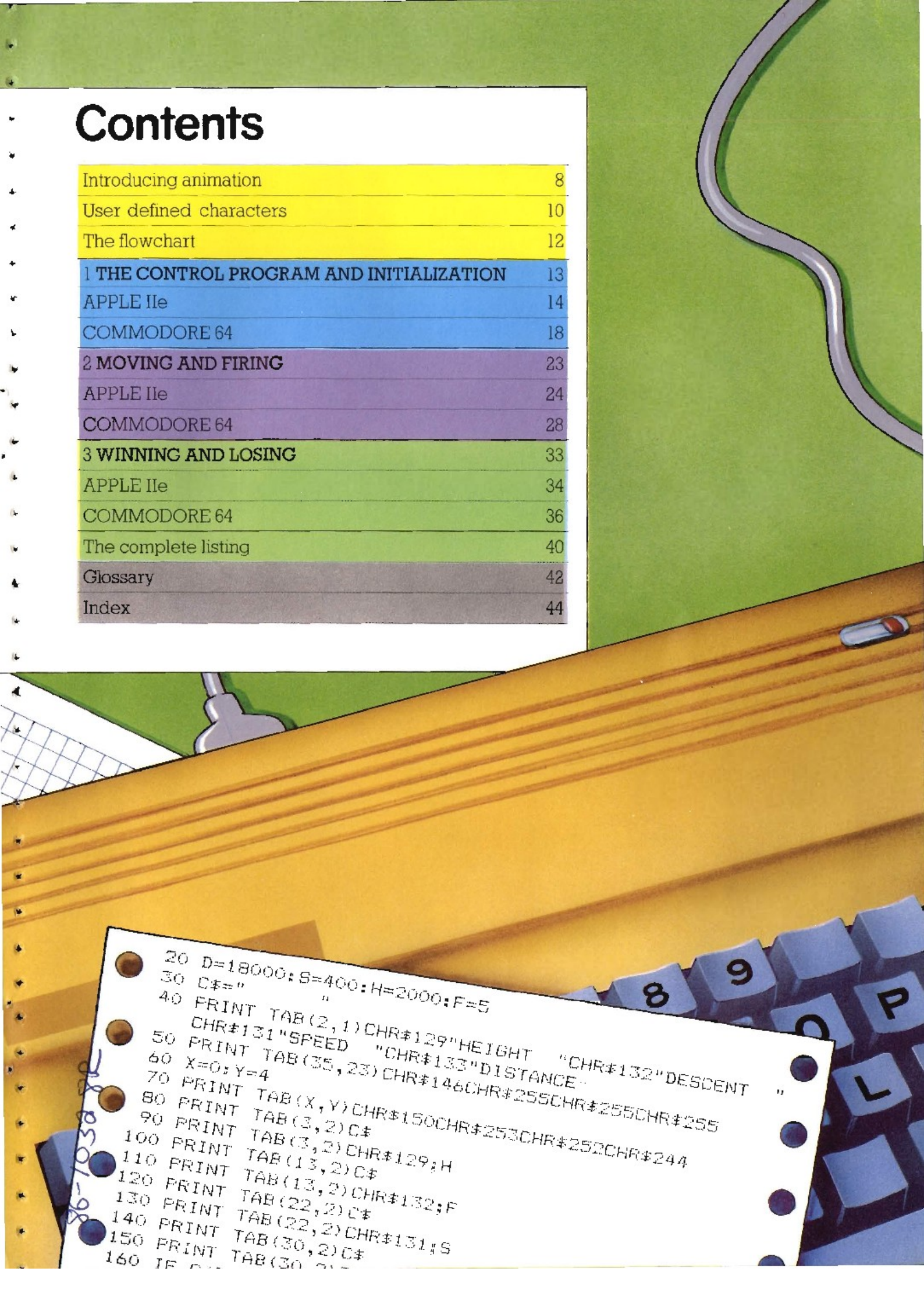
GLOUCESTER PRESS
NEW YORK · TORONTO · 1985

Foreword

If you have a computer, then you are almost certain to have played a Space Invaders game of one kind or another. In this book, you'll find a program that allows you to create an "invaders" game of your own, in full color. The program is given for the Apple IIe and Commodore 64 computers.

An invaders game relies upon moving graphics to be effective. The principle is similar to those used in cartoons - old characters are erased from the screen before being printed in a new position, giving the impression of movement. The program has been broken down to its logical stages, and there is a running text which explains how each section works. You will find, for example, just how it is that the computer "knows" when an invader has been shot by one of your missiles. Writing a program in this way not only makes it easier for you and other people to understand, it also makes it run more quickly and reduces the possibility of errors. At various stages you can test the sections of program that you've keyed in – even the smallest error can mean that the program doesn't work. If something does go wrong check back through the listing very carefully. Looking for "bugs" in a program is like playing a detective game – the clues are there for you to spot them.

Contents



Introducing animation

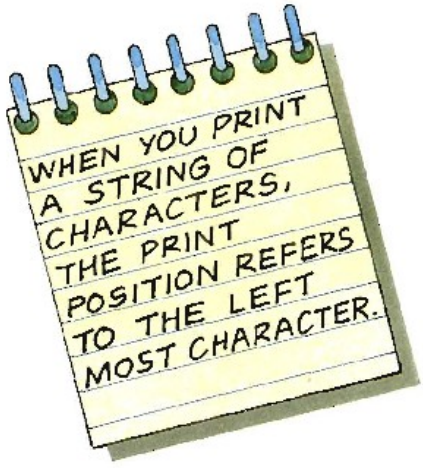
One of the simplest forms of computer animation is achieved by printing a figure on the screen, erasing it and then quickly printing the same figure again in a slightly different position. Repeating this action a number of times gives the illusion of the figure moving rapidly across the screen. This short program demonstrates the effect.

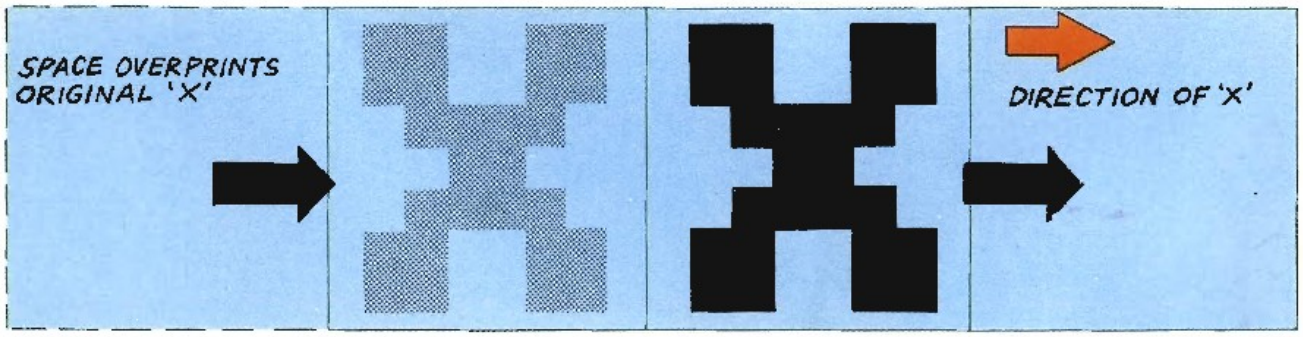
```
10  HOME
20  FOR I = 1 TO 38
30  VTAB 5: HTAB I: PRINT " X"
40  FOR D = 1 TO 100: NEXT D
50  NEXT I
```

```
10 PRINT"L":PRINT"[RIGHT]"
20 FOR I=1 TO 38
30 PRINTSPC(I)"[3] X"
40 FOR D=1 TO 200:NEXT D
50 NEXT I
```

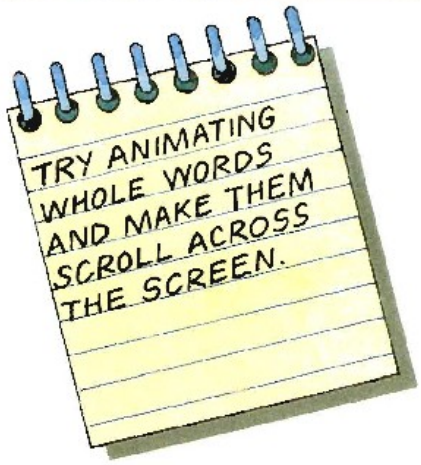
The letter X is moved across the screen using a **FOR . . . NEXT** loop, with **I** as its variable – the changing value of **I** gives different screen coordinates for printing "X". Notice the space either side of the X. When the letter is moved one space along to the right, the spaces either side move with it. The blank space on the left will overprint the old X and effectively erase it from the screen.

In line 40 a short delay loop is introduced, making the computer count to 100 (Apple IIe) or 200 (Commodore) before it prints the next "X". Without the delay loop, the computer would perform the whole operation so quickly it would be impossible to follow. Delete line 40 and see what happens.





Commodore Sprites

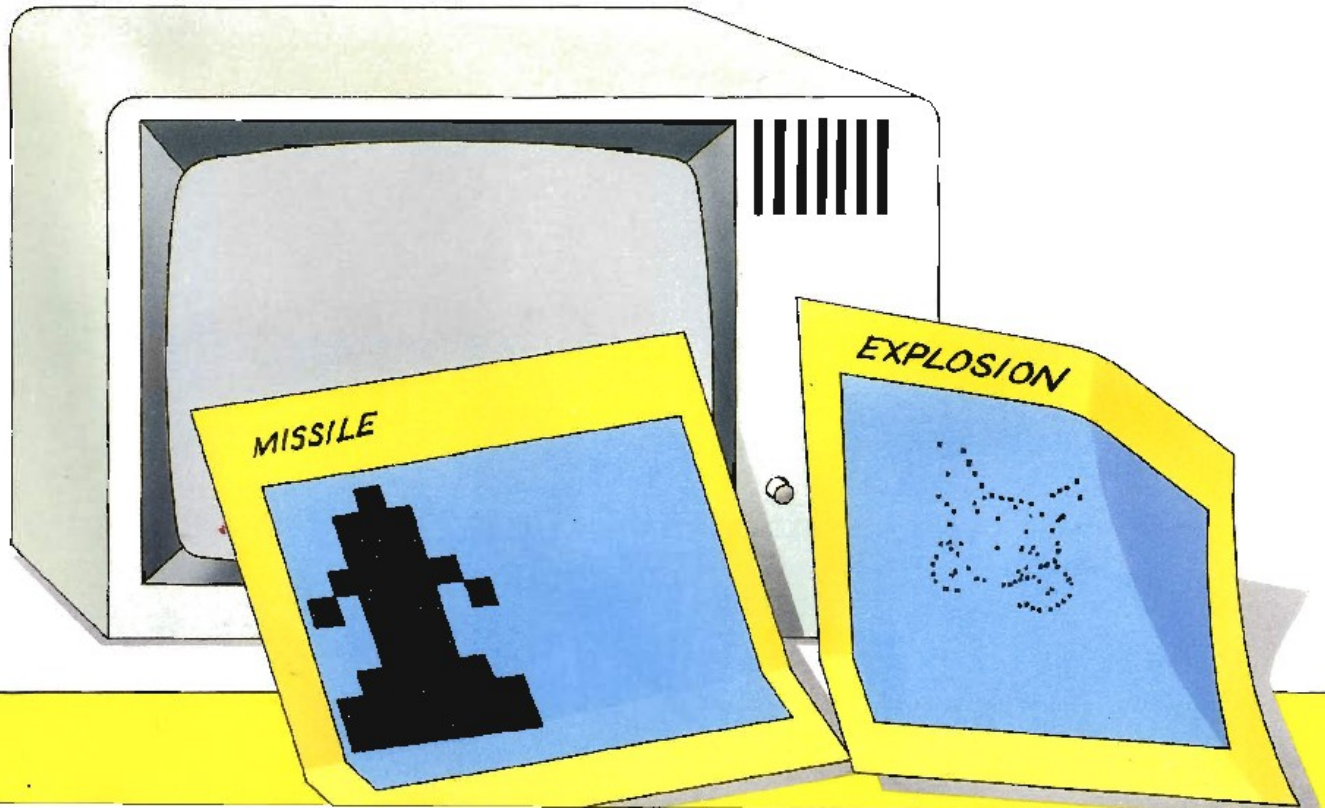


In addition to the move-and-blank technique described opposite, the Commodore has its own special moving graphics called *sprites*.

Sprites do not have to be blanked out and reprinted many times to make them appear to move. Instead, you will have to use the special *registers* in the table below. Later in the book you will see how, by **POKE**ing numbers into these registers, sprites for missiles and explosions can be created and controlled.

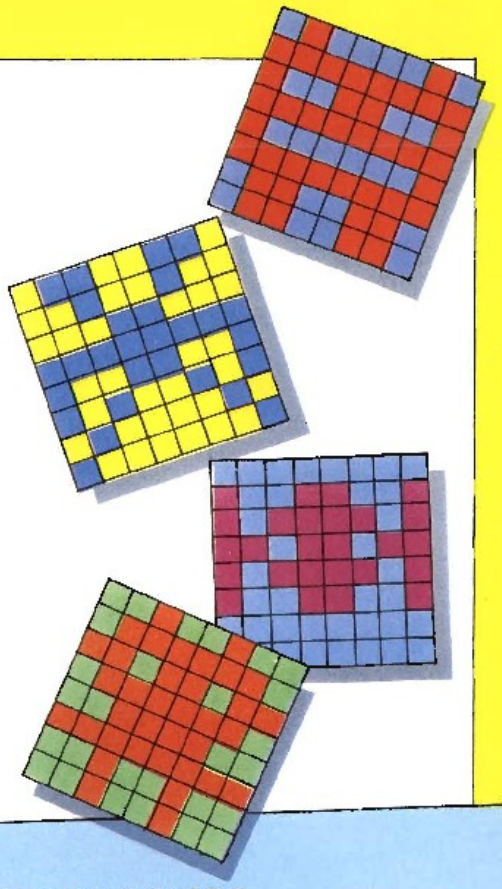
Sprites are a complex, but very flexible feature of the Commodore and there are many more special sprite registers. Unfortunately, in this book there isn't room to much more than show you a small percentage of the possibilities open to you, but with practise, you will be able to create stunning animated graphics.

Memory Location	Name
832	start of sprite data area
53248	X location register
53249	Y location register
53259	sprite enabling register
53264	X register in VIC (video) chip
53269	Y register in VIC (video) chip
53287	color register - sprite one
53289	color register - sprite two



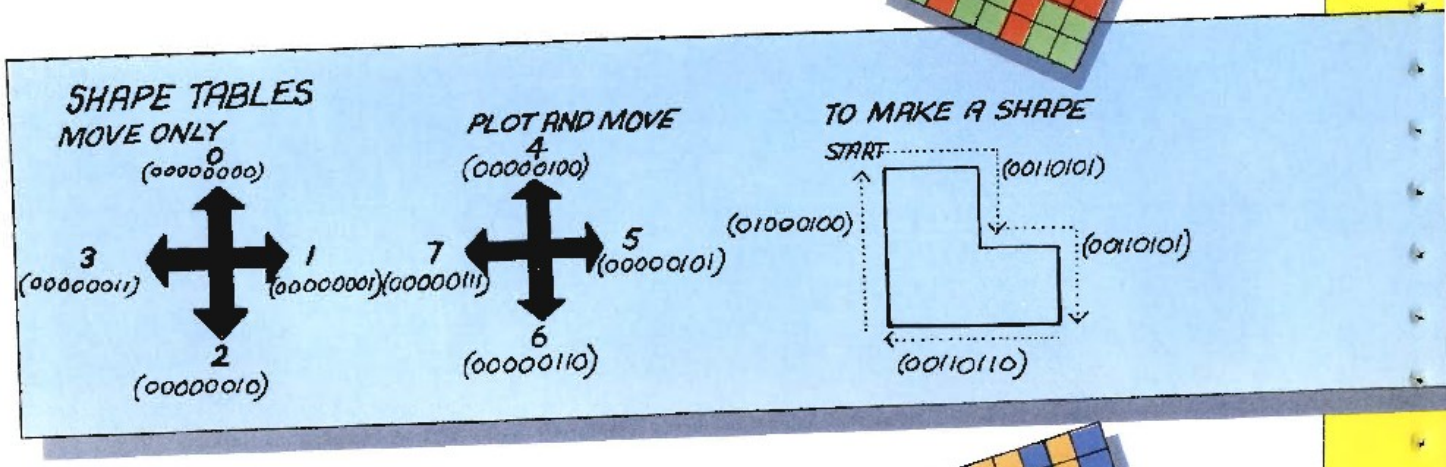
User defined characters

Games can be made to look far more exciting and professional by defining your own user defined graphics, which can be animated using the techniques you will see later in the book. In the Commodore program you will be using sprites.

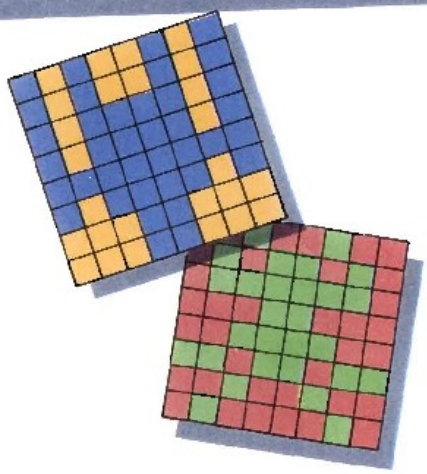


APPLE IIe

Defining graphics on the Apple involves using *shape tables*. These can be very difficult to use until you become familiar with them. The screen is made up of many dots (or *pixels*), and you can instruct your Apple to draw shapes by joining these dots together. There are eight instructions, numbered in binary 000 to 111. Instructions 000 to 011 are called *move only* operations, which tell the computer to move a pointer either up, down, left or right one pixel on the screen.



With a *move only* instruction you can move an imaginary pointer around the screen without drawing anything. Instructions 100 to 111 are *plot and move* operations – these draw as well as move. Starting from a defined point on the screen, you use the move only instructions to move the pointer to where you wish to commence drawing, before using the plot and move instructions to draw it.

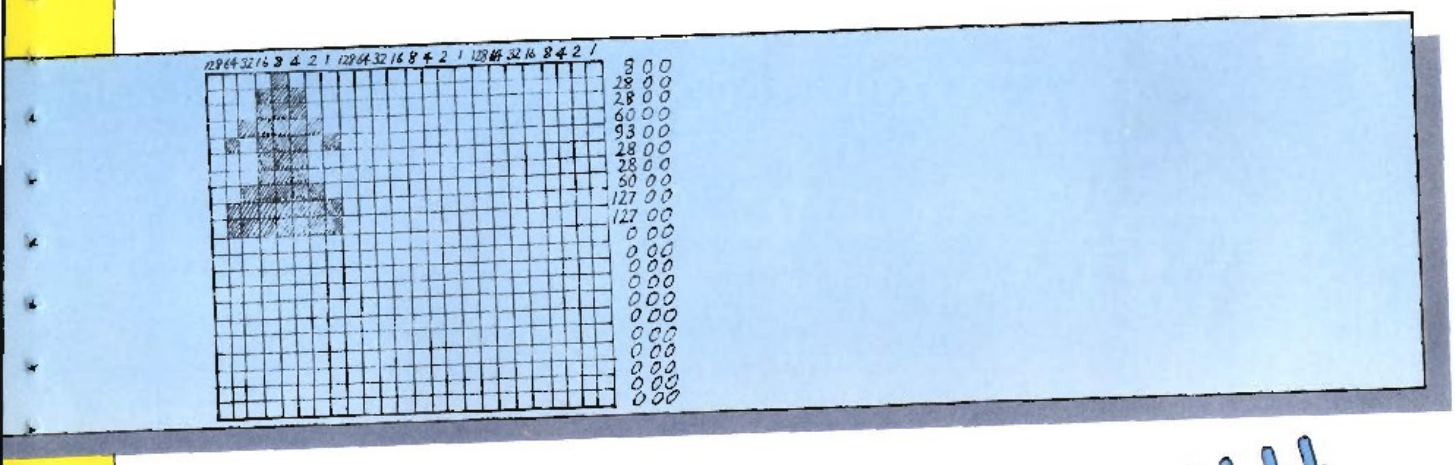


The computer doesn't understand these instructions as they stand as 3-bit (binary digit) numbers. They first need to be collected together into 8-bit numbers, and then converted to decimal numbers (the kind of numbers humans count in). Each 8-bit number contains two shape table operations, usually preceded by two zeroes. You can convert from binary to decimal by imagining each bit as a column representing a decimal number – starting from the right, the columns represent 1, 2, 4, 8, 16, 64, and 128 - and adding the columns together

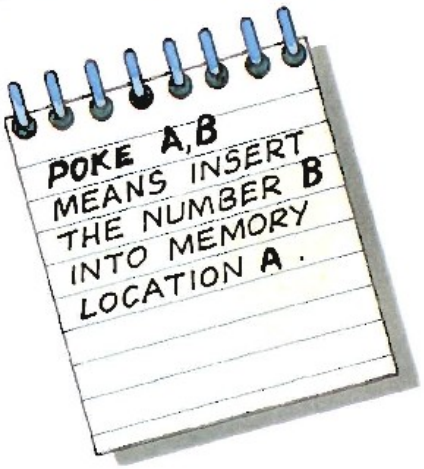
COMMODORE

Computer graphics screens consist of many small dots (or *pixels*). Design your sprite on graph paper using a 24- by 21-pixel grid. You can draw any shape you like, shading squares that you want the computer to color. It understands that each dot can have the value of 0 or 1. If the dot is set to 1, then the computer interprets this as a filled dot. If the dot is set to 0, the computer takes the dot as blank. These 1s and 0s are known as binary digits (or bits). There are eight bits to a byte, and the computer stores each byte in a memory location. Sprites are defined by **POKE**ing 63 bytes into a special area of memory. **POKE** tells the computer to put a number into a specific memory location.

Sprites can be any size up to 24 by 21 pixels. The diagram below shows how the missile sprite used later on in the book is drawn and how the **DATA** is calculated.

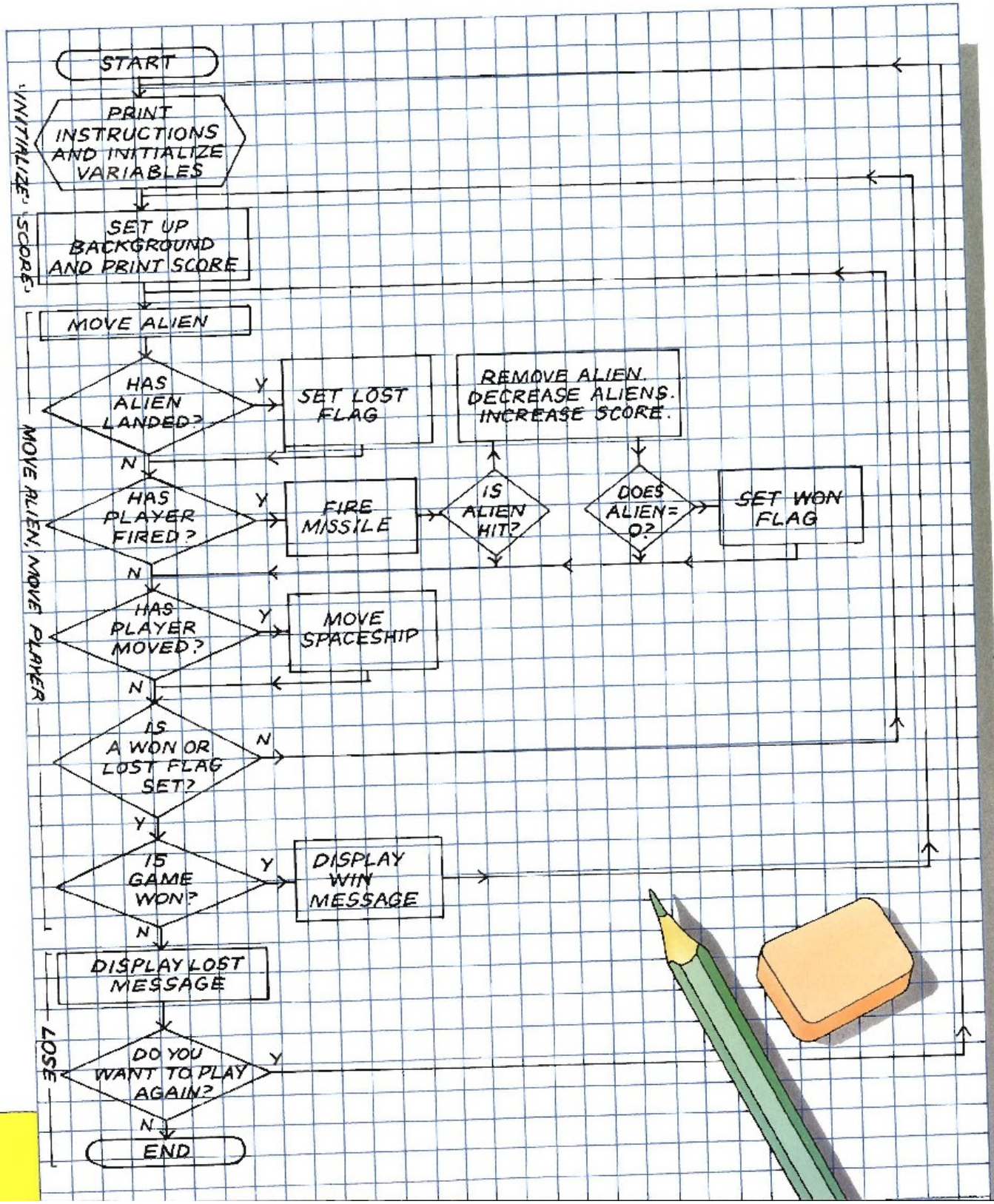


Setting up a sprite involves a number of steps, using some special registers in the Commodore. First, **DATA** must be **POKE**d into an area of memory starting at location **832**. The **DATA** has to be converted from the eight bit binary numbers into decimal numbers (the numbers that humans understand, and programs use) before it can be put into a program. You can do this by adding up the numbers at the top of the columns in the diagram above. Up to eight sprites can be set up at the same time, each needing their own *sprite pointer* so the computer knows which sprite you want to use. Next, the sprite needs to be turned on using the *sprite enabling register* (location **53248+21**). The computer needs to know what color to make the sprite, so a color code must be **POKE**d into the *color register* (location **53248+39** for sprite 1, location **53248+40** for sprite 2, and so on). Finally, the sprite needs to be positioned on screen, using X and Y *location registers*. You will see how the sprites for the missile and explosion used in the game are set up in this program later in the book.



The flowchart

To plan the Invaders game, it's a good idea to draw up a flowchart. This is a convenient way of setting out the series of logical steps that define your program. It consists of different shaped boxes, representing decisions, actions and so on, connected by "flowlines". The flowchart is used to create a CONTROL PROGRAM. This is the core of the program, which calls up other sections of the program whenever they are required.



THE CONTROL PROGRAM AND INITIALIZATION

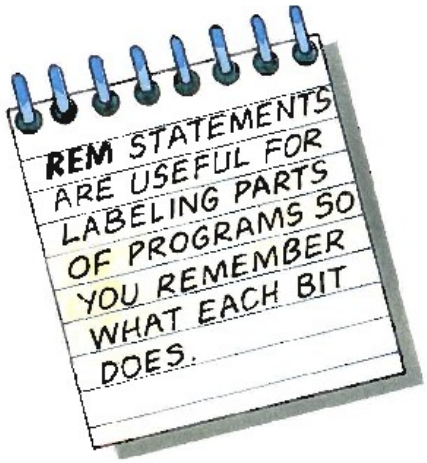
The control program shows how the program is structured, using a series of logical steps called subroutines on the Apple and Commodore machines. These are grouped together on the flowchart to show how the control program has been constructed. The first section of the program then displays instructions for the game on the screen.

```
   YOU ARE A LONE SPACE PILOT
PROTECTING THE PLANET EARTH. IN
A FEW MOMENTS YOU WILL BE UNDER
ATTACK BY ALIENS FROM THE PLANET
VARGON
YOUR MISSION IS TO PREVENT ANY
VARGONIAN SHIP FROM LANDING.

  'Z' MOVES STARFIGHTER LEFT
  'X' MOVES STARFIGHTER RIGHT

  PRESS SPACE TO FIRE MISSILE

PRESS ANY KEY TO ENGAGE ENEMY
```

By following the flowchart on page 12 and breaking the program into subroutines placed in a logical order, you can arrive at the main program for the game, as shown below. Once you have a main program like this, all you have to do is write each subroutine in turn.

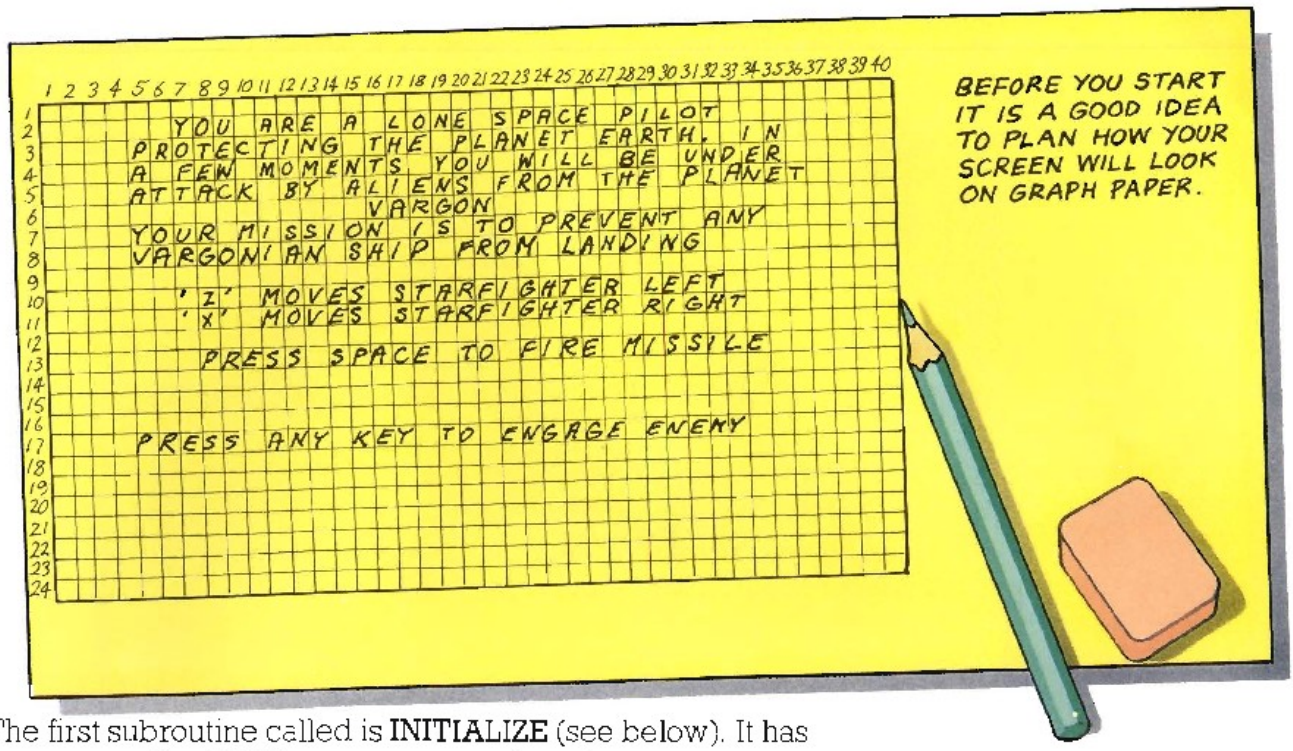
```
5  REM    INVADERS
10  GOSUB 8000: REM    INITIALIZE
20  GOSUB 7000: REM    SCREEN
30  GOSUB 1000: REM    ALIEN
40  GOSUB 2000: REM    PLAYER
50  IF WIN = 1 OR LOST = 1 THEN  GOTO 70
60  GOTO 30
70  IF WIN = 1 THEN  GOSUB 5000: GOTO 20
80  GOSUB 6000: REM    LOSE
90  END
```

The first line of a program usually gives its title, by way of a **REM**ark – see line 5. **REM**arks help anyone see what is happening in a program. At the beginning of a game you should give the player instructions, and set all the variables to give the player instructions, and set all the variables to their initial values. Line 10 does this. The colon (:) allows separate statements to be placed on the same line. The subroutine which plots the stars and prints the score is called by line 20.



The next four lines, 30 to 60, form a repeated loop which is the heart of the program. A randomly selected alien is moved one position down the screen in line 30. Line 40 allows the player to move and fire a rocket at the aliens. Next comes an **IF . . . THEN** test in Line 50. **IF** either of these conditions are true, **THEN** the computer jumps out of the loop to line 70. Otherwise, line 60 sends the computer back to line 30, ready to repeat the move/fire process. **WIN** and **LOST** are simply "flags" which can be set true or false (1 or 0) to tell the computer how the game has ended. Line 70 tests if the **WIN** flag is set. **IF** it is, **THEN** the **WIN** subroutine is called, using the instruction **GOSUB 5000**. The computer **GO**es **TO** line 20 for a fresh screen of aliens, and a new game. If **WIN** isn't set, the program calls the **LOSE** subroutine, starting at line 6000. All the subroutines follow later in the book.

The **END** statement may be placed anywhere that is convenient. However, it is good practice to place the **END** at the end of the control program. This gives a neat finish and makes the program more readable.

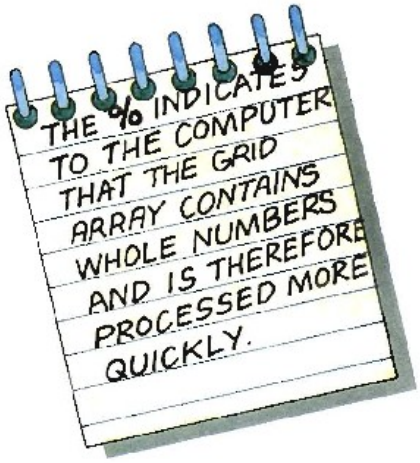


The first subroutine called is **INITIALIZE** (see below). It has been put at line 8000 because, every time the computer looks for a subroutine it starts at line 1, looking at each line in turn. Subroutines which you use frequently should be put at the start of the program. In line 8010 we clear the text screen, and bring the cursor to the home position (to the top left hand corner of the screen) using call **TEXT**. **VTAB** means vertical tab, or move down a certain number of text lines on screen. **HTAB** is exactly the same but for horizontal movement. In line 8020, the computer is instructed to start **PRINT**ing down two lines, and across seven characters.

```
8000  REM     INITIALIZE
8010  TEXT : HOME
8020  VTAB 2: HTAB 7: PRINT "YOU ARE A LONE SPACE
PILOT"
8030  HTAB 5: PRINT "PROTECTING THE PLANET EARTH. IN"
8040  HTAB 5: PRINT "A FEW MOMENTS YOU WILL BE UNDER"
8050  HTAB 5: PRINT "ATTACK BY ALIENS FROM THE PLANET"
8060  HTAB 16: PRINT "VARGON."
8070  HTAB 5: PRINT "YOUR MISSION IS TO PREVENT ANY"
8080  HTAB 5: PRINT "VARGONIAN SHIP FROM LANDING."
8090  VTAB 10: HTAB 7: PRINT "'Z' MOVES STARFIGHTER
LEFT"
8100  VTAB 12: HTAB 7: PRINT "'X' MOVES STARFIGHTER
RIGHT"
8110  VTAB 14: HTAB 7: PRINT "PRESS SPACE TO FIRE
MISSILE"
8120  VTAB 18: HTAB 5: PRINT "PRESS ANY KEY TO ENGAGE
ENEMY ";
```

```
8130  GOSUB 9000
8140 LOST = 0:WIN = 0:SC = 0:AHIT = 0:HITSTAR = 0:XP
=16:HEIGHT = 0:INVADERS = 10:ALIEN = INVADERS
8150 ALIENS = INVADERS
8160  DIM X(INVADERS): DIM Y(INVADERS): DIM GD%(35,19)
8170  FOR A = 1 TO INVADERS
8180 X(A) = A * 3:Y(A) = HEIGHT
8190  NEXT A
8200  GET R$
8210  RETURN
```

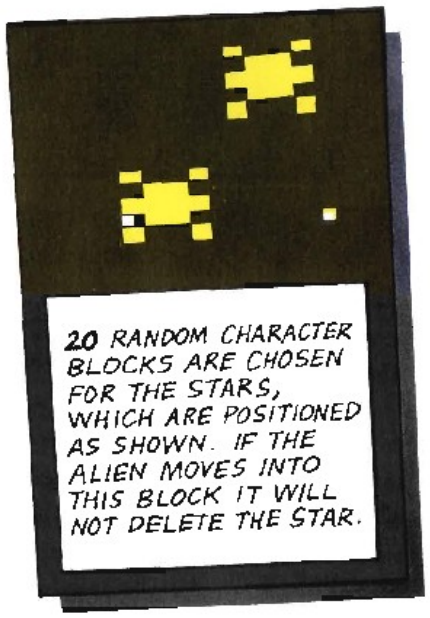
As the player is reading the instructions, the computer can be getting on with initializing the variables used in the game. In line 8140 a number of flags are set. **LOST** and **WIN** are set to zero since the game has yet to begin. **SC** represents the number of aliens that have been shot down. **AHIT** (alien hit) and **HITSTAR** are two flags which check if a missile has collided with an alien or a star. **XP** is the position of the player's spaceship along the screen's X-axis. **HEIGHT** is the Y position of the space invaders when they start towards the top of the screen. In Line 8150 the number of **ALIENS** is set by **INVADERS**. This extra variable is used to make it very easy to change the number of space invaders at the start of the game. Once **INVADERS** has been altered the initial value of **ALIENS** will automatically alter throughout the program. Each space invader is given its own label by **DIM**ensioning two one-dimensional arrays in line 8160. **DIMX(INVADERS)** reserves memory space for the column position of each space invader, and **DIMY(INVADERS)** does the same for the row position. For example, **X(2)** refers to the second space invader working from the left of the screen. **GD%** is an integer (whole number) array which holds the location of all the characters. Line 8170 starts a loop which sets the initial positions of the **INVADERS**. Line 8180 enters the X and Y position of each invader. The X position is determined by multiplying the alien's number by three, and the Y position by **HEIGHT**.




```
9000  FOR X = 7676 TO 7804
9010  READ A
9020  POKE X,A
9030  NEXT X
9040  DATA  5,0,12,0,25,0,33,0,80,0,86,0,73,169,63,46,
53,63,46,53,63,22,45,5,00,73,45,109,58,223,27,55,109,7
3,53,255,219,55,109,73,53,63,63,63,55,54,37,108,73,53,
53,45,00,41,77,9,245,59,63,159,45,45,45,255,219,55,45,
45,45
```

```
9050  DATA  245,59,63,159,46,108,73,53,63,00,73,145,58
,46,5,00,13,77,13,182,58,223,36,220,147,46,61,22,109,1
13,43,107,57,4,193,36,4,193,193,32,36,223,255,27,255,5
1,150,173,191,82,118,182,41,69,41,5,193,193,1,00
9060  RETURN
```

Five shapes are defined for the game – the missile, the player's spaceship, the alien, the star and the explosion. The **DATA** in lines 9040 and 9050 is **POKE**d into memory by the **FOR . . . NEXT** loop between lines 9000 and 9030. In the **DATA** the first two numbers indicate how many shapes are being defined (in this case five). The following ten numbers form five pairs, telling the computer where the data for each shape begins.



The **SCREEN** subroutine **DRAW**s a random background of stars before the game begins or when the player is skilful enough to shoot all the aliens. Line 7010 clears the screen and sets the screen to Page 1 of high resolution graphics (**HGR**). In line 7020, **POKE – 16301,0** allows you to use the bottom four lines of the screen for text. **POKE**ing locations **232** and **233** tells the computer where to find the shape table information for the stars and **SCALE** sets their size.

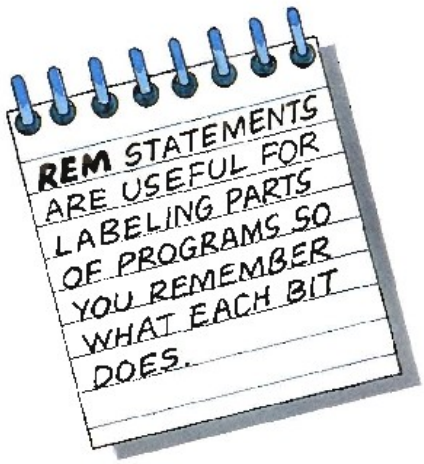
```
7000  REM  SCREEN
7010  HGR : HOME
7020  POKE  - 16301,0: POKE 232,252: POKE 233,29:
SCALE= 1
7030  FOR S = 1 TO 20
7040 DF =  INT ( RND (1) * 35):GH =  INT ( RND (1) *
 9)
7050 GD%(DF,GH) = 1
7060  HCOLOR= 7: DRAW 4 AT 8 * DF,8 * GH
7070  NEXT S
7080  VTAB 22: HTAB 10: PRINT "ALIENS DESTROYED=";SC
7090  HCOLOR= 2: DRAW 2 AT 8 * XP,8 * 19
7100  RETURN
```

There is a **FOR . . . NEXT** loop between lines 7030 and 7070 which plots 20 stars on the screen. Line 7040 uses your machine's random number generator to decide where to put the stars. Each time the computer goes through the **FOR . . . NEXT** loop an X and a Y coordinate are chosen. Line 7050 stores the value one in the element in **GD%** which corresponds to the star position. Line 7090 **DRAW**s the spacecraft at the bottom of the screen, ready to start firing.

By following the flowchart on page 12 and breaking the program into subroutines placed in a logical order, you can arrive at the main program for the game, as shown below. Once you have a main program like this, all you have to do is write each subroutine in turn.

```
5 REM INVADERS
10 GOSUB 8000: REM INITIALIZE
20 GOSUB 7000: REM SCREEN
30 GOSUB 1000: REM ALIEN
40 GOSUB 2000: REM PLAYER
50 IF WIN=1 OR LOST=1 THEN GOTO 70
60 GOTO 30
70 IF WIN=1 THEN GOSUB 5000: GOTO 20
80 GOSUB 6000: REM LOSE
90 END
```

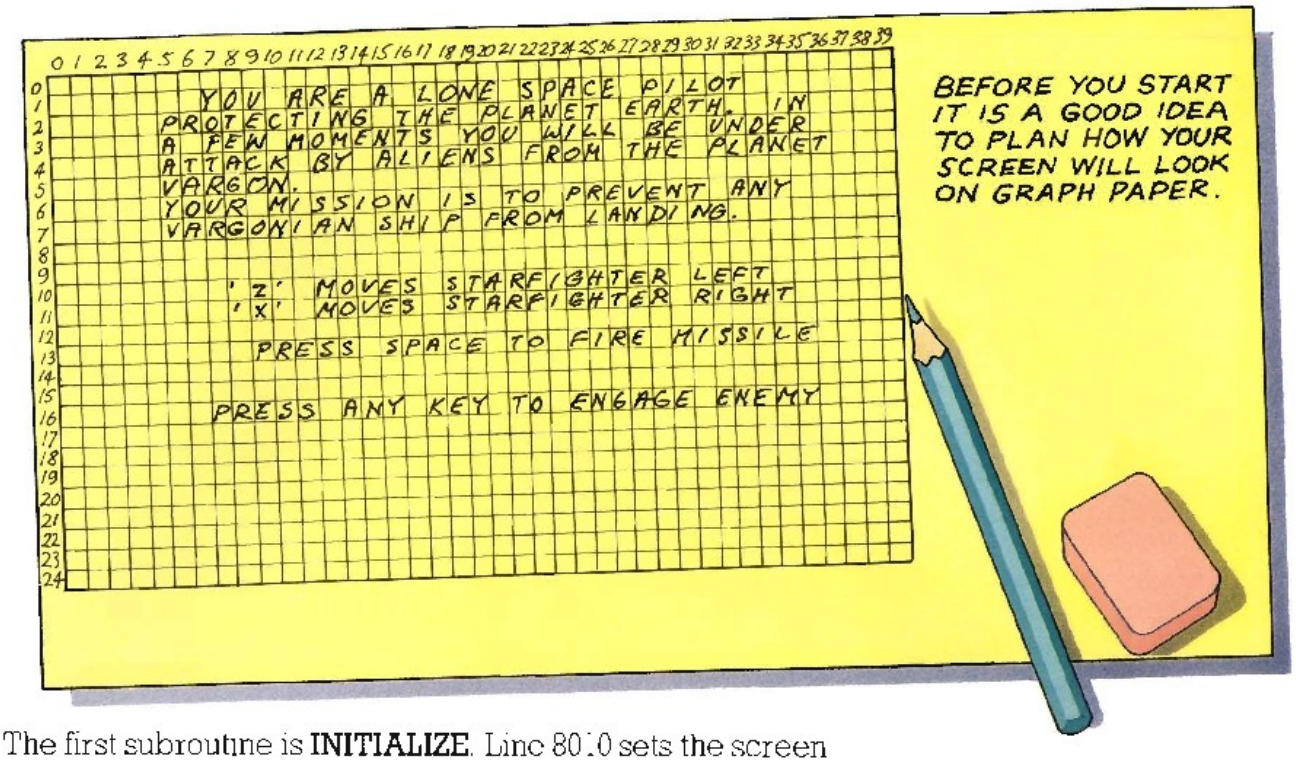
The first line of a program usually gives its title, by way of a **REM**ark – see line 5. **REM**arks help anyone see what is happening in a program.



At the beginning of a game you should give the player instructions, and set all the variables to their initial values. Line 10 does this. The colon (:) allows separate statements to be placed on the same line. The subroutine which plots the stars and prints the score is called by line 20. The next four lines, 30 to 60, form a repeated loop which is the heart of the program.

A randomly selected alien is moved one position down the screen in line 30. Line 40 calls a subroutine allowing the player to move and fire a rocket at the aliens. Next comes an **IF . . . THEN** test in Line 50. **IF** either of these conditions are true, **THEN** the computer jumps out of the loop to line 70. Otherwise, line 60 sends the computer back to line 30, ready to repeat the move/fire process. **WIN** and **LOST** are simply "flags" which can be set true or false (**1** or **0**) to tell the computer how the game has ended. Line 70 tests if the **WIN** flag is set. **IF** it is, **THEN** the **WIN** subroutine is called, using the instruction **GOSUB 5000**. The computer immediately **GO**es **TO** line 20 for a fresh screen of aliens, and a new game. If **WIN** isn't set, the program calls the **LOSE** subroutine, starting at line 6000.

The **END** statement may be placed anywhere that is convenient. However, it is good practice to place the **END** at the end of the control program. This gives a neat finish and makes the program more readable.

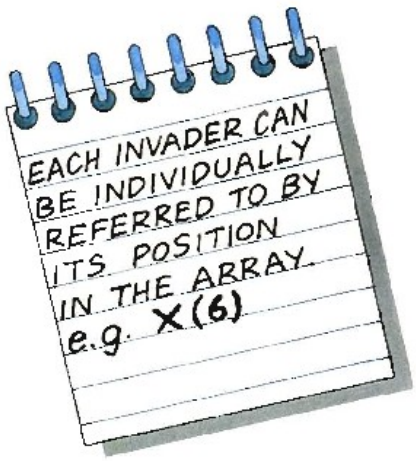


The first subroutine is **INITIALIZE**. Line 8010 sets the screen colors by **POKE**ing locations **53280** (the border) and **53281** (the background) with zero (black). **POKE 650,255** sets the keyboard to auto-repeat so the spacecraft moves smoothly, and the heart-shaped control character clears the screen. Lines 8020 to 8120 contain **SPC** (SPaCe) commands, to position the display. The control characters used determine the text color.

```
8000 REM INITIALIZE
8010 POKE 53280,0: POKE 53281,0: POKE 650,255:
     PRINT"■"
8020 PRINTSPC(5)"■YOU ARE A LONE SPACE PILOT"
8030 PRINTSPC(4)"PROTECTING THE PLANET
     EARTH. IN"
8040 PRINTSPC(4)"A FEW MOMENTS YOU WILL
     BE UNDER"
8050 PRINTSPC(4)"ATTACK BY ALIENS FROM
     THE PLANET"
8060 PRINTSPC(4)"VARGON"
8070 PRINTSPC(4)"YOUR MISSION IS TO
     PREVENT ANY"
8080 PRINTSPC(4)"VARGONIAN SHIP FROM
     LANDING."
8090 PRINTSPC(6)"■■'Z' MOVE S STARFIGHTER
     LEFT"
8100 PRINTSPC(6)"'X' MOVES STARFIGHTER RIGHT"
8110 PRINTSPC(7)"■PRESS SPACE TO FIRE LASER"
8120 PRINTSPC(5)"■■PRESS ANY KEY TO ENGAGE
     ENEMY"
```

```
8130 LOST=0: WIN=0: SC=0: AHIT=0: HITSTAR=0: XP=20:
     HEIGHT=1: INVADERS=10
8140 ALIENS=INVADERS
8150 DIM X(INVADERS): DIM Y(INVADERS): DIM GRID%(40,21)
8160 FOR A=1 TO INVADERS
8170 X(A)=A*4-2: Y(A)=HEIGHT
8180 GRID%(X(A),Y(A))=2
8190 NEXT A
8200 GOSUB 8500
8210 R$=""
8220 GET R$: IF R$="" THEN 8220
8230 RETURN
```

As the player is reading the instructions, the computer can be getting on with initializing variables. In line 8130 a number of flags are set. **LOST** and **WIN** are set to zero since the game has yet to begin. **SC** represents the number of aliens that have been shot down. **AHIT** is a flag which is used to check if a missile has collided with an alien. **XP** is the position of the player's spaceship along the screen's X-axis. **HEIGHT** is the Y position of the space invaders. In line 8140 the number of **ALIENS** is set by **INVADERS**. If you alter the value of **INVADERS**, the initial value of **ALIENS** will automatically alter throughout the program. Each invader is given its own label by **DIM**ensioning two arrays in line 8160. **DIMX (INVADERS)** reserves memory space for the column position of each space invader, and **DIMY (INVADERS)** does the same for the row position. Line 8160 starts a **FOR . . . NEXT** loop which sets the initial positions of the **INVADERS**. Line 8170 enters the X and Y position of each invader.




```
8500 REM SPRITE DEFINITION
8510 FOR N=0 TO 62: READ Q: POKE 832+N,Q: NEXT:
     REM MISSILE
8520 FOR N=0 TO 62: READ Q: POKE 896+N,Q: NEXT:
     REM EXPLOSION
8530 V=53248
8540 POKE 2040,13
8550 POKE 2041,14
8560 POKE V+39,2
8570 POKE V+40,7
8580 DATA 8,0,0,28,0,0,28,0,0,62,0,0,93,0,0,28,0,0,
     28,0,0,62,0,0,127,0,0,127,0,0
8590 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
     0,0,0,0,0,0,0,0,0,0,0,0,0
8600 DATA 0,0,0,136,0,17,2,0,64,1,0,128,16,165,8,129,
     0,129,33,0,132,18,165,72
```

```
8610 DATA 12,20,48,18,165,72,0,92,0,0,92,0,72,165,19,
     48,90,12
8620 DATA 72,165,18,132,0,33,124,0,129,8,165,16,128,
     0,1,64,0,2,17,0,136
8630 RETURN
9000 PRINT"■";: FOR I=1 TO Y: PRINT"■";: NEXT: RETURN
```

On pages 9 and 11 you saw how Commodore sprites are created and used in theory. This routine sets up the missile and explosion sprites needed for the game using the Commodore's special registers. Line 8510 **POKE**s the 63 bytes into memory, starting at location 832. Line 8530 makes **V=53248** – this is the lowest numbered memory location and every higher numbered location only needs be numbered as **V** plus a small number. The sprite pointers are set up in lines 8540 and 8550. Next, the sprite needs to be turned on using the *sprite enabling register* (location **V+21**).

The computer needs to know what color to make the sprite, so line 8560 **POKE**s a color code into a *color register* (location **V+39** for sprite one, location **V+40** for sprite 2, and so on). In this case the code is 2, which is the equivalent to red sprite graphics.

```
7000 REM SCREEN
7010 POKE V+21,0: PRINT"■"
7020 FOR S=1 TO 40
7030 X=INT(RND(1)*40): Y=INT(RND(2)*21): PRINT"■":
     GOSUB 9000:PRINTTAB(X)"■."
7040 GRID%(X,Y)=1
7050 NEXT S
7060 PRINT"■":PRINTSPC(10)"ALIENS DESTROYED=";SC
7070 RETURN
```

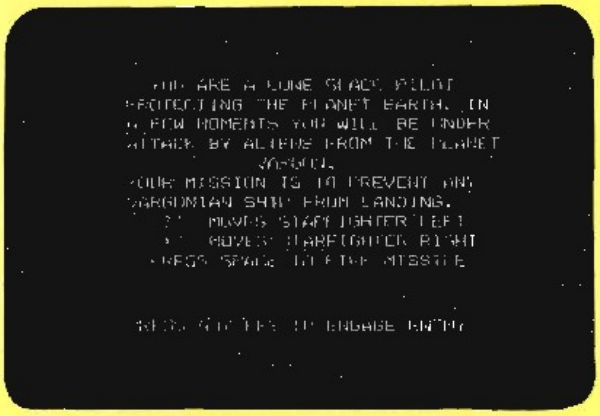
Line 20 of the control program instructs the computer to **GOSUB 7000**. This subroutine draws a random background of stars. Line 7010 clears the screen, ready for the background. There is a **FOR . . . NEXT** loop between lines 7020 and 7050 which plots 40 stars on the screen. Line 7030 uses your machine's random number generator to decide where to put the stars. Each separate part of an array is called an element, and a two-dimensional array such as **GRID%** can be used to represent a map by making each element correspond to a grid square on the map. Line 7040 stores the value one in the element in **GRID%** which corresponds to the star position that has just been chosen.

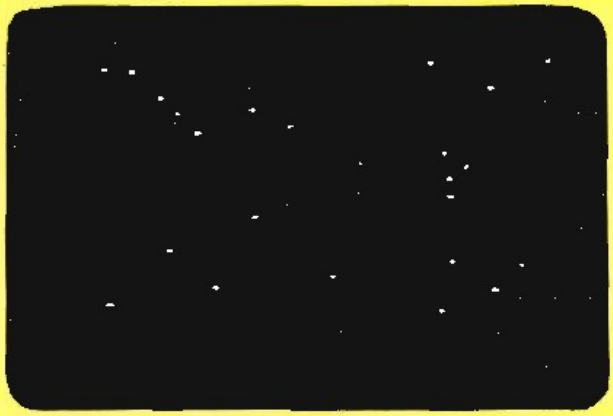
Testing your program

At this point it is a good idea to test the program. Follow the instructions in the box below and you should see the title screen. Pressing any key will bring up the second screen. If the program doesn't work, or something peculiar happens check through your work very carefully. Even tiny errors can cause the whole program to crash.

APPLE IIe 1. TYPE A DUMMY LINE **25 END** AND HIT RETURN ↵
2. TYPE **RUN** AND HIT RETURN ↵

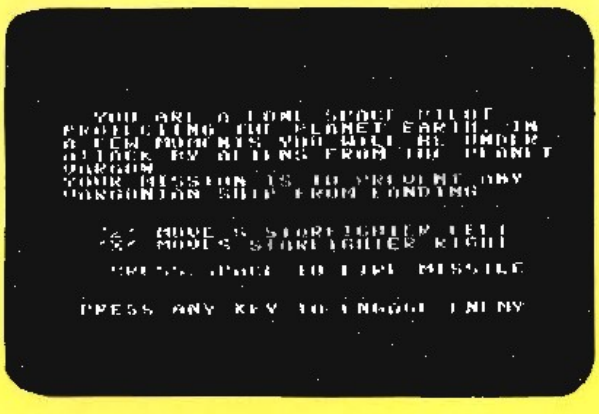
YOU SHOULD NOW SEE THE TITLE SCREEN SHOWN LEFT. PRESSING ANY KEY WILL ALLOW THE PROGRAM TO CONTINUE, AND THE STAR BACKGROUND WILL BUILD UP, SHOWN RIGHT. DELETE THE DUMMY LINE 25 BEFORE PROCEEDING TO THE NEXT SECTION.





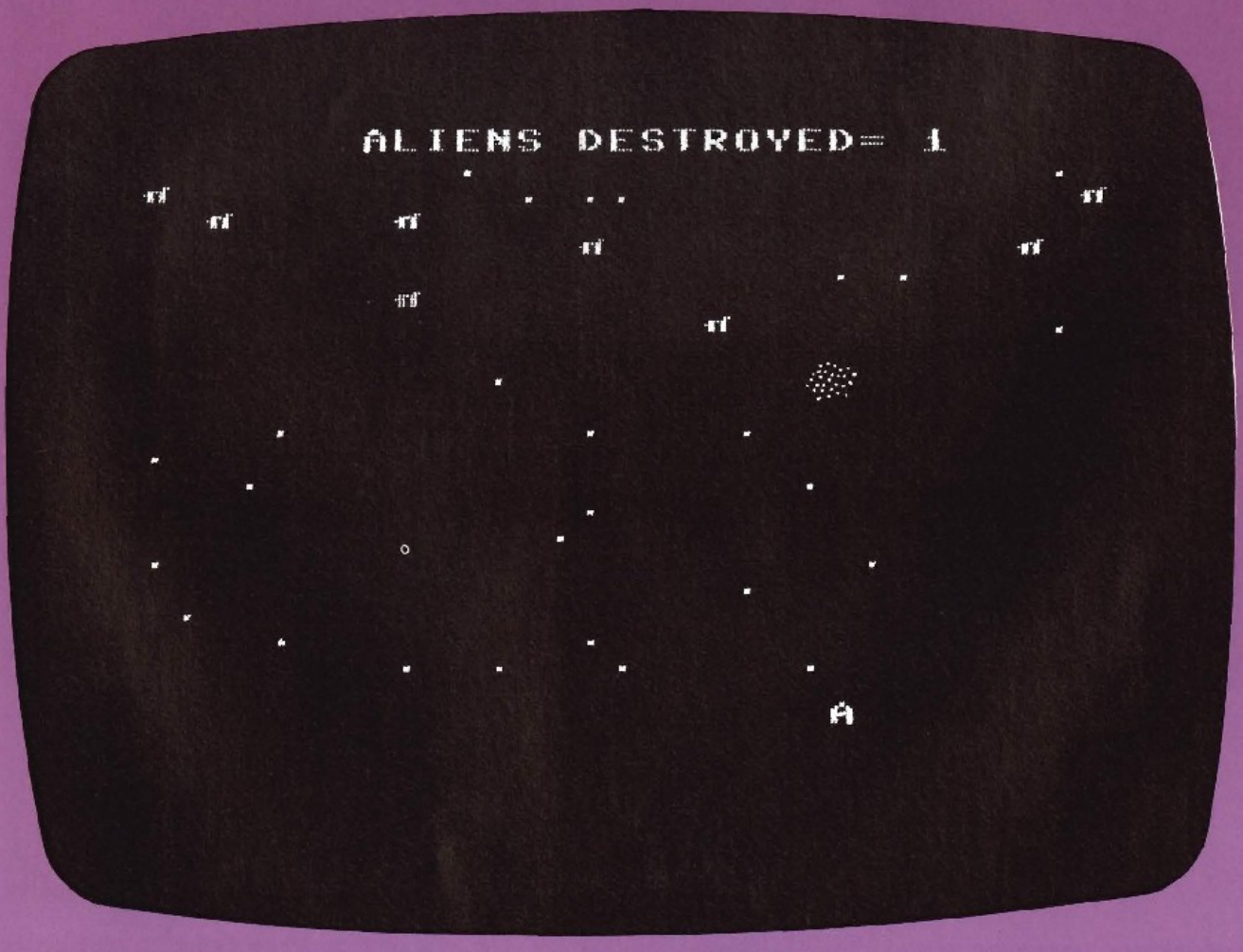
COMMODORE 1. TYPE A DUMMY LINE **25 END** THEN HIT RETURN
2. TYPE **RUN** THEN HIT RETURN

YOU SHOULD NOW SEE THE TITLE SCREEN SHOWN LEFT. PRESSING ANY KEY WILL ALLOW THE PROGRAM TO CONTINUE, AND THE STAR BACKGROUND WILL BUILD UP, SHOWN RIGHT. DELETE THE DUMMY LINE 25 BEFORE PROCEEDING TO THE NEXT SECTION.

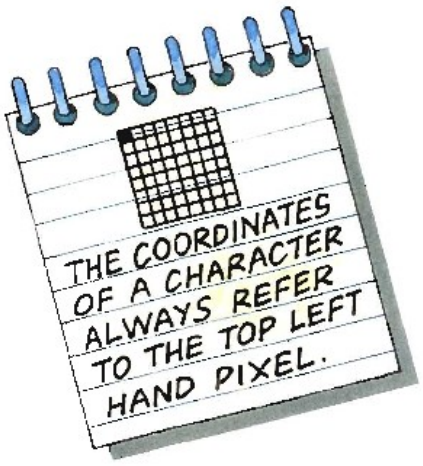


MOVING AND FIRING

In this section of program the aliens move at random down and across the screen. It also contains the instructions which allow the player to move his or her spaceship and to fire missiles to shoot the aliens down. A running score of the number of aliens hit is given at the top of the screen.
Good Luck!



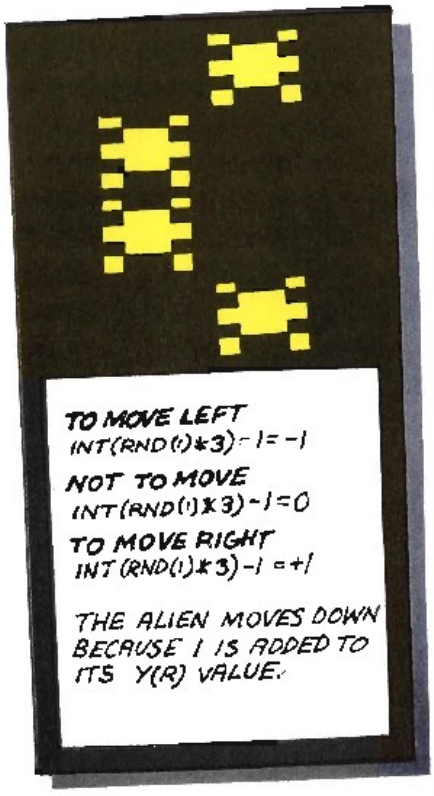
The **ALIEN** subroutine given below has a low number because it is the most frequently used of the subroutines. The subroutine moves the aliens. It works by choosing an alien at random, and blanking it out by **DRAW**ing the character in black on the screen.



The alien remains invisible until the alien's next position has been calculated by the computer. Once the new position has been calculated, the alien is **DRAW**n on the screen in yellow. All this happens very quickly and gives the impression of smooth movement, and works a little like a cartoon film.

```
1000  REM   ALIEN
1010 R =  INT ( RND (1) * INVADERS) + 1: IF Y(R) = 22
THEN  RETURN
1020 GD%(X(R),Y(R)) = GD%(X(R),Y(R)) - 2
1030  HCOLOR= 0: DRAW 3 AT 8 * X(R),8 * Y(R)
1040 X(R) = X(R) +  INT ( RND (1) * 3) - 1
1050 Y(R) = Y(R) + 1
1060  IF X(R) > 33 THEN X(R) = 33
1070  IF X(R) < 1 THEN X(R) = 1
1080  IF Y(R) = 19 THEN LOST = 1
1090  HCOLOR= 5: DRAW 3 AT 8 * X(R),8 * Y(R)
1100 GD%(X(R),Y(R)) = GD%(X(R),Y(R)) + 2
1110  RETURN
```

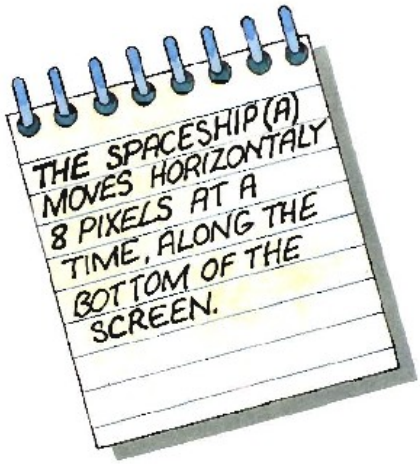
Line 1010 selects a random whole number between one and the value of **INVADERS**. The value of **R** identifies a particular alien. Line 1010 also checks the alien's vertical coordinate. If it is 22, then it is off-screen, and is "dead" – the subroutine ends with a **RETURN** – see the **HIT** subroutine later on. If a "living" alien has been chosen, line 1030 **DRAW**s an alien in black at **Y(R), X(R)** making it disappear. New coordinates are calculated in lines 1040 and 1050, using the **RND** function. A random number – either minus one, zero or one -- is added to the current value of **X(R)** to alter the column position, and the value of **Y(R)** is increased by one. The effect of this is to move the alien's new position one space left and down, or one space down, or one space right and down. Lines 1060 and 1070 limit the value of **X(R)**, and make sure that the alien can never be off the screen. The next line, 1080, tests the value of **Y(R)**. If it is equal to 19, then the alien has reached the bottom of the screen, and the player has lost. The **LOST** flag is therefore set. In line 1090 the alien is **DRAW**n in yellow at its new position. Finally, **GD%** is updated by adding 2 to the value in the element corresponding to the alien's X and Y coordinates.



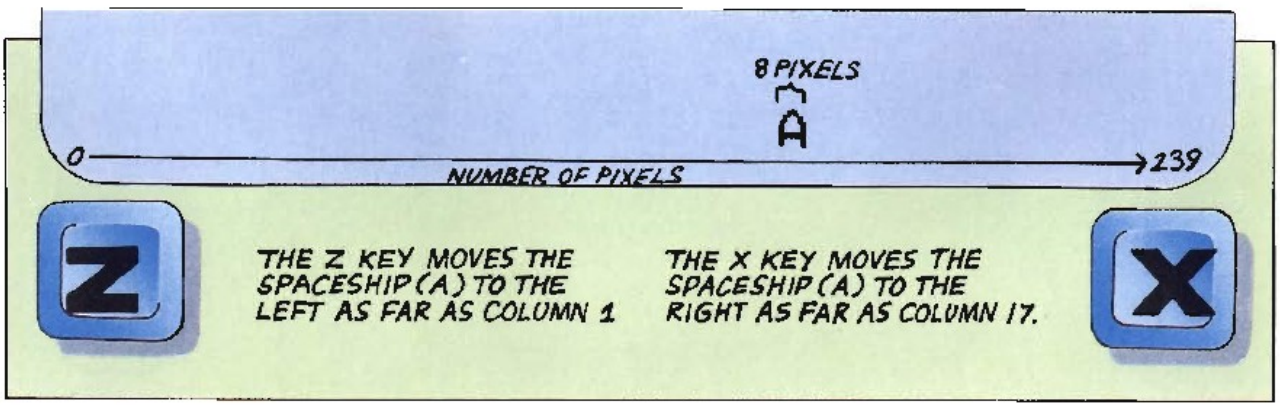
```
2000  REM       PLAYER
2010  I = PEEK ( - 16384): IF I < 128 THEN 2070
2020  I$ = CHR$ (I - 128)
2030  POKE - 16368,0
2040  HCOLOR= 0: DRAW 2 AT 8 * XP,19 * 8
2050  IF (I$ = "Z" OR I$ = "z") AND XP > 1 THEN XP =
XP - 1
2060  IF (I$ = "X" OR I$ = "x") AND XP < 33 THEN XP =
XP + 1
2070  HCOLOR= 2: DRAW 2 AT 8 * XP,19 * 8
2080  IF I = 160 THEN GOSUB 3000
2090  RETURN
```

The first of the next two subroutines allows you to control the player's spacecraft, and to fire missiles. The second subroutine animates missiles and checks for hits.

In line 2010, **PEEK**ing (– **16384**) looks at the keyboard. The key press is stored in **I$**. The Z key will move the player's spacecraft left and the X key moves it to the right.

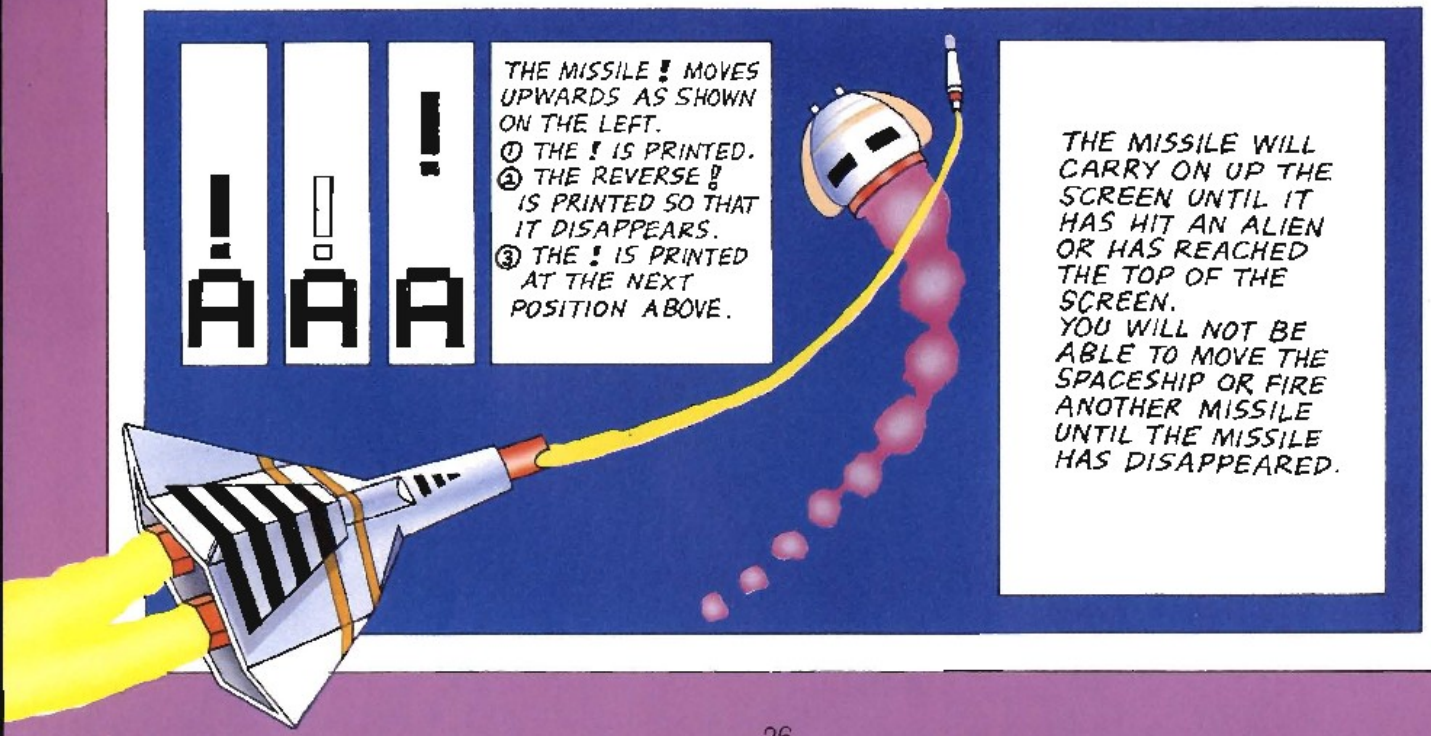


Line 2040 **DRAW**s the spacecraft in black. Lines 2050 and 2060 control the movement of the spacecraft. **IF** the Z key has been pressed **AND** the spacecraft's position is greater than one, **THEN** the value of **XP** is decreased by one character position. Using the < (greater than) and the > (less than) operators in this way ensures that the spacecraft can never be off screen. The expression **XP=XP-1** is common in computer programming – all it means is that the variable is altered by the amount specified. In this case, **XP** is decreased by one. The spacecraft is held in shape number two and is re**DRAW**n in purple by line 2070 – this happens even if the spacecraft hasn't moved. Line 2080 checks if the player has pressed the space bar (in which case **I=160**), and calls the fire routine if appropriate. Finally, line 2090 **RETURN**s control to the main program.



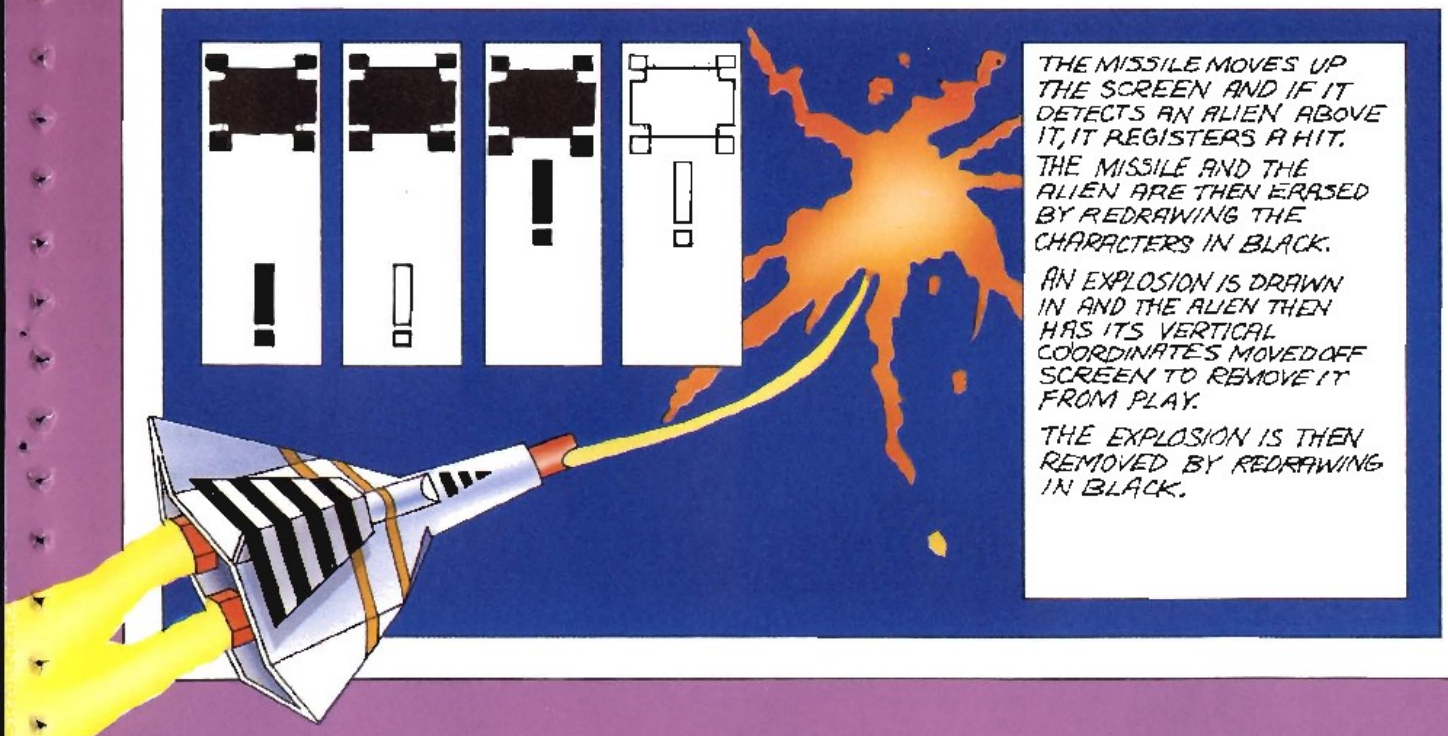
```
3000  REM   FIRE
3010 XM = XP * 8
3020  FOR YM = 17 TO 0 STEP -1
3030  HCOLOR= 2: DRAW 1 AT XM,YM * 8
3040  IF GD%(XP,YM) > 1 THEN  GOSUB 4000
3050  HCOLOR= 0: DRAW 1 AT XM,YM * 8
3060  IF GD%(XP,YM) = 1 THEN  HCOLOR= 7: DRAW 4 AT XM,
YM * 8
3070  NEXT
3080  RETURN
```

This routine animates the missile. There is a **FOR . . . NEXT** loop between lines 3020 and 3070, which is set up with **STEP-1**. This is a little different from a normal **FOR . . . NEXT** loop because, instead of counting up, the loop counts down in steps of one. Whenever you use a **FOR . . . NEXT** loop you can specify whichever **STEP** suits your program best. In this case starting at **YM=17**, the missile is moved up the screen in steps of one line until it reaches the top line (**YM=0**). The missile is **DRAW**n in purple by line 3030 – **HCOLOR=2** means **DRAW** the shape in purple. In the next line there is a check to see if that position is already occupied by an alien – is there a hit? If the value held in **GD%** for that location is greater than one, the **HIT** routine starting at line 4000 is called. The missile is blanked out in line 3050 by **DRAW**ing it in black at the same screen position. **GD%** is checked again in line 3060, but this time the check is for a star. If the missile has gone over a star, the star will not reappear unless it is **DRAW**n back on screen. Line 3060 takes the star's coordinates and **DRAW**s it in again in white.

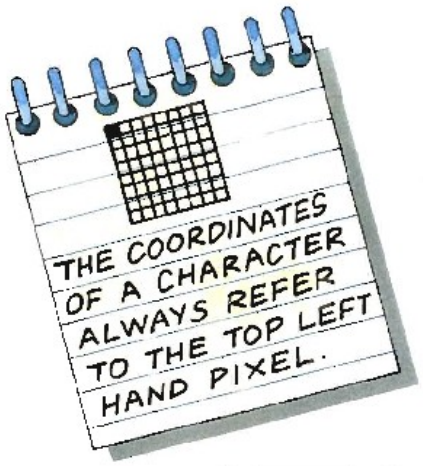


```
4000  REM       HIT
4010  FOR A = 1 TO INVADERS
4020  IF X(A) <  > XP OR Y(A) <  > YM THEN  GOTO 4080
4030  HCOLOR= 0: DRAW 3 AT 8 * X(A),8 * Y(A)
4040  HCOLOR= 0: DRAW 1 AT 8 * X(A),8 * Y(A)
4050  HCOLOR= 7: DRAW 5 AT 8 * X(A),8 * Y(A)
4060 GD%(X(A),Y(A)) = GD%(X(A),Y(A)) - 2
4070 Y(A) = 22:A = INVADERS
4080  NEXT A
4090 ALIEN = ALIEN - 1:SC = SC + 1:AHIT = 0
4100  IF ALIEN = 0 THEN WIN = 1
4110  VTAB 22: HTAB 27: PRINT SC
4120  HCOLOR= 0: DRAW 5 AT 8 * XP,8 * YM
4130 YM = 0
4140  RETURN
```

The **FOR . . . NEXT** loop between lines 4010 and 4080 checks each of the invaders in turn to see which one has been hit. The alien which has been hit is **DRAW**n in black in line 4030, and the missile is **DRAW**n in black in line 4040. In line 4050 an explosion is **DRAW**n in the position where the alien has been hit. Having destroyed the alien, line 4060 updates **GD%**. The alien's position is set to 22 so that it will be skipped over in line 1010. The loop is exited by setting **A** equal to **INVADERS**. One is subtracted from the number of **ALIEN**s in line 4090, and the score (**SC**) increased to give you a running score display. The **AHIT** flag is reset to zero. If line 4100 finds that no more **ALIEN**s remain, the **WIN** flag is set. The score is displayed by line 4110, and finally the explosion is **DRAW**n over in black in line 4120.

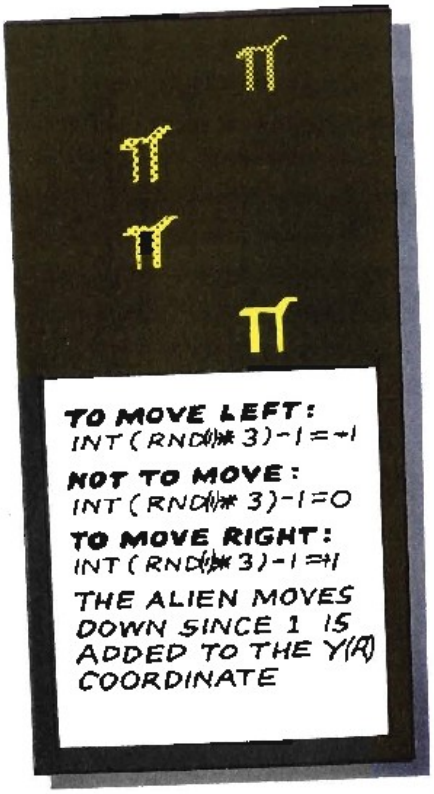


The **ALIEN** subroutine given below has a low number because it is the most frequently used of the subroutines. The subroutine moves the aliens – represented by a pi sign. It works by choosing an alien at random, and then blanking it out by printing the character in black on the screen. The alien remains invisible until the alien's next position has been calculated by the computer. Once the new position has been calculated, the alien is printed back on the screen in yellow. All this happens very quickly and gives the impression of the figure moving rapidly across the screen, and works a little like a cartoon film.




```
1000 REM ALIEN
1010 R=INT(RND(1)*INVADERS)+1: IF Y(R)=22 THEN RETURN
1020 Y=Y(R): X=X(R): GOSUB 9000: PRINTSPC(X)" ";
1030 GRID%(X,Y)=GRID%(X,Y)-2
1040 IF GRID%(X(R),Y(R))=1 THEN GOSUB 9000:
     PRINTSPC(X)"■";
1050 X(R)=X(R)+INT(RND(1)*3)-1
1060 Y(R)=Y(R)+1
1070 IF X(R)>36 THEN X(R)=36
1080 IF X(R)<1 THEN X(R)=1
1090 IF Y(R)=21 THEN LOST=1
1100 Y=Y(R): X=X(R): GOSUB 9000: PRINTSPC(X)"■";
1110 GRID%(X,Y)=GRID%(X,Y)+2
1120 RETURN
```

Line 1010 selects a random whole number between one and the value of **INVADERS**. The value of **R** identifies a particular alien. Line 1010 checks the vertical coordinate of the alien. If it is 22, then it is off-screen, and is "dead" – the subroutine ends with a **RETURN** instruction. What is happening here will become more clear when the **HIT** subroutine is described later on. If a "living" alien has been chosen, line 1020 prints a space in black at **Y(R)**, **X(R)**. New coordinates are calculated in line 1050, using the **RND** function. A random number – either minus one, zero or one – is added to the current value of **X(R)** to alter the column position, and the value of **Y(R)** is increased by one. The effect of this is to move the alien's new position one space left and down, or one space down, or one space right and down. Lines 1070 and 1080 set limits to the value of **X(R)**, and make sure that the column position can never be off the screen. The next line, 1090, tests the value of **Y(R)**. If it is equal to 21, then the alien has reached the bottom of the screen, and the player has lost. The **LOST** flag is therefore set. In line 1100 the alien is printed in yellow at its new position. Finally, **GRID%** is updated by adding 2 to the alien's X and Y coordinates.

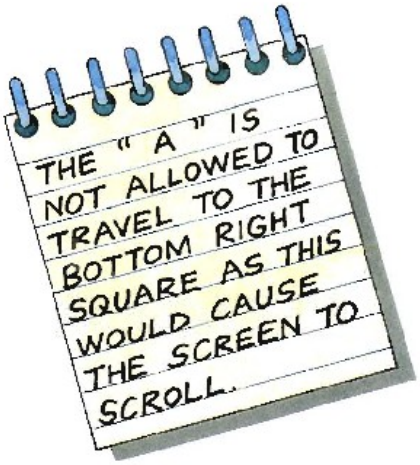


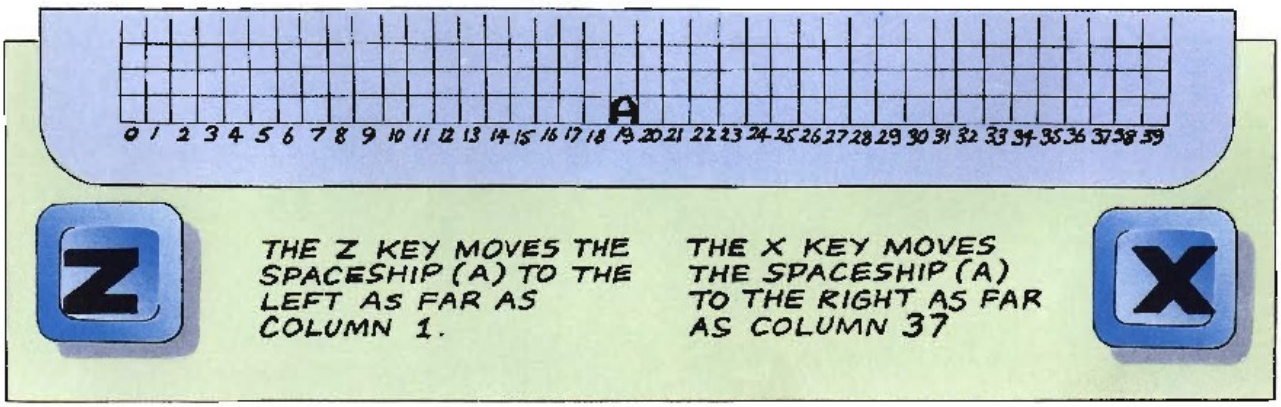
```
2000 REM PLAYER
2010 GET L$
2020 IF L$=" " THEN GOSUB 3000: REM FIRE
2030 IF L$="Z" AND XP>1 THEN XP=XP-1
2040 IF L$="X" AND XP<37 THEN XP=XP+1
2050 Y=22: GOSUB 9000: PRINTSPC(XP-1)" A ";
2060 RETURN
```

The first of the next two subroutines allows you to control the player's ship, and to fire missiles. The player's ship can move left and right across the bottom of the screen. The second subroutine animates missiles and checks for hits.

In real-time games such as this one the player needs to be able to interact with what is happening on the screen. It is quite easy to write a routine which allows the player to use the keyboard to control graphics.

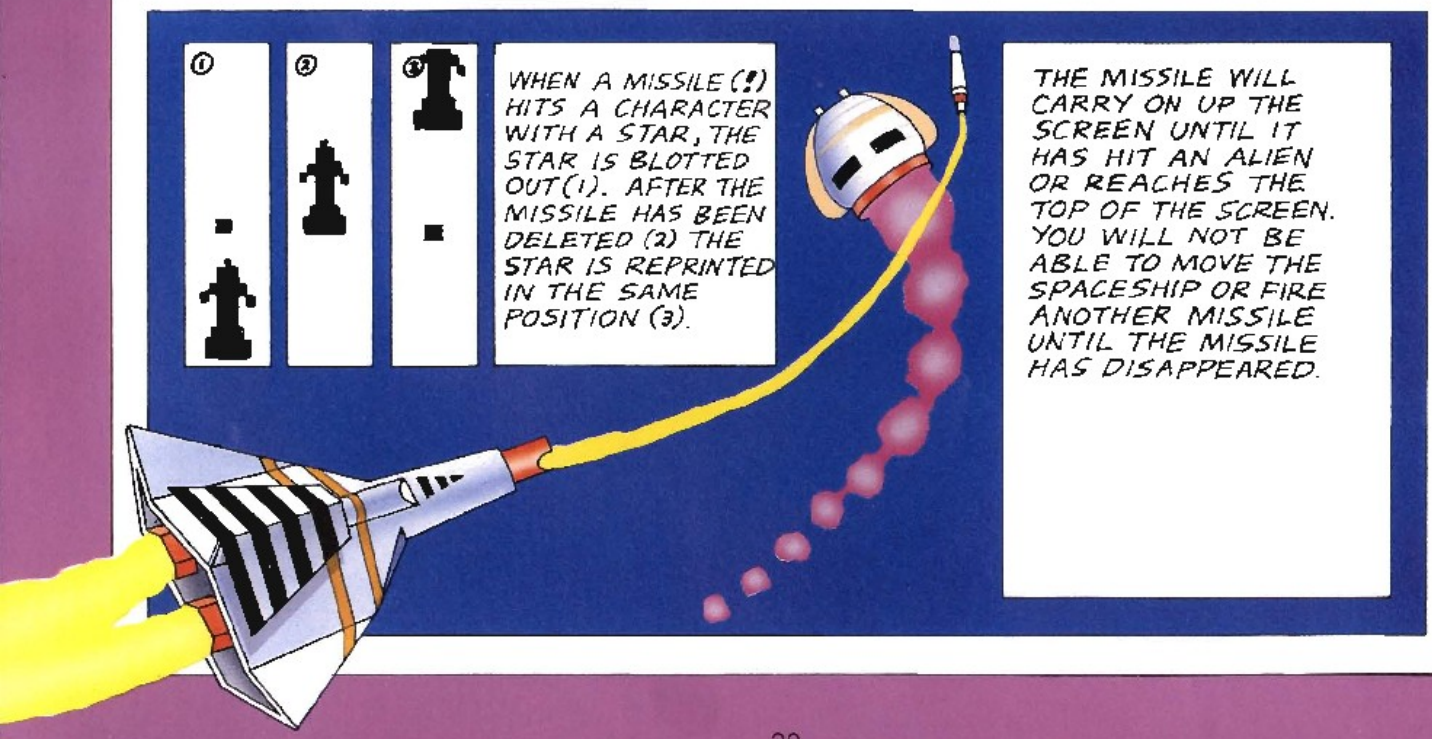
In line 2010 **GET** looks at the keyboard to see if a key has been pressed, and stores the result in **L$**. Now you should test for the keys you are interested in. The next line tests **L$**, and **IF** it is a space (the player has pressed the space bar), **THEN** the instruction is to **GOSUB 3000**, which is the missile firing subroutine – defined below. Lines 2030 and 2040 control the movement of the spacecraft. **IF L$** is **Z** (the Z key has been pressed) **AND** the spacecraft's position is greater than one, **THEN** the value of **XP** is decreased by one. Using the > (greater than) and the < (less than) operators in this way ensures that the spacecraft can never be off screen. The expression **XP=XP−1** is common in computer programming – all it means is that the variable is altered by the amount specified. In this case, **XP** is decreased by one. The player's spaceship is represented by the letter A, which is **PRINT**ed on screen in a string of characters (" A "), between two blank spaces. The spaces over-print and hence erase the old character A when the spaceship is moved left or right.





```
3000 REM FIRE
3010 X=(XP*8)+24
3020 FOR YM=21 TO 1 STEP -1
3030 Y=(YM*8)+50
3040 IF X<256 THEN POKE V,X: POKE V+16,0
3050 IF X>255 THEN POKE V,X-255: POKE V+16,1
3060 POKE V+1,Y: POKE V+21,1
3070 IF GRID%(XP,YM)>1 THEN GOSUB 4000
3080 NEXT
3090 POKE V+21,0
3100 RETURN
```

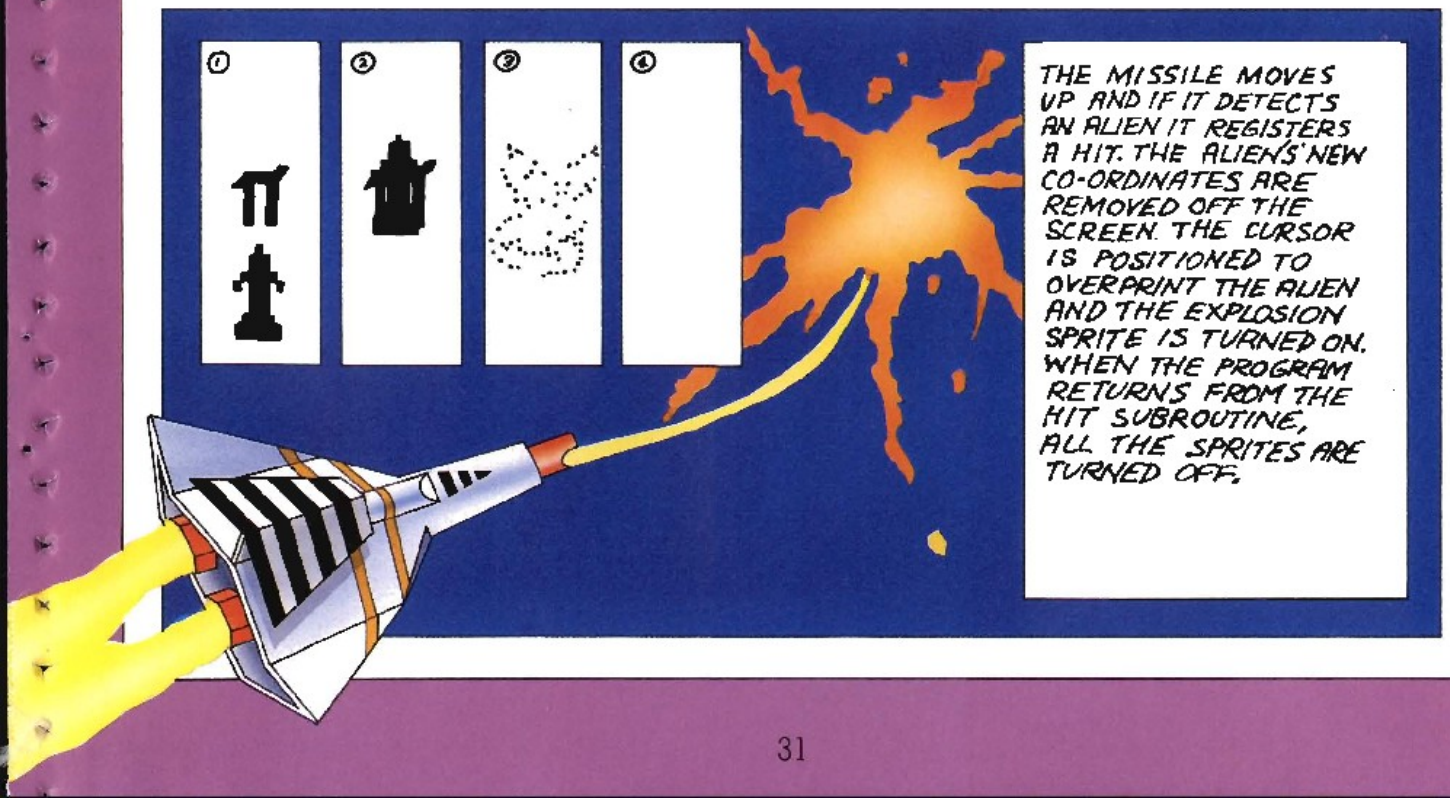
The subroutine above is the **FIRE** routine, which animates the missile. There is a **FOR . . . NEXT** loop with **STEP −1** between lines 3020 and 3080, which moves the missile from the bottom of the screen to the top. The sprite is displayed on screen using the X and Y location registers and registers in the VIC (video) chip. In the program these registers are referred to as **V**, **V+1**, **V+16** and **V+21**. Unfortunately, you cannot **POKE** the screen coordinates directly into these registers. The value for the X location register is calculated in line 3010. If X is less than 256 it is **POKE**d directly into **V**, along with zero into **V+16**, in line 3040. If X is greater than 255 (the largest value a single memory location can contain) line 3050 subtracts 255 from X. Line 3030 calculates a new value for the Y location register. Line 3060 **POKE**s the Y location register, and switches on the sprite using the sprite enable register (**V+21**). A hit on an alien by the missile is detected in line 3070 and the program is directed to the **HIT** subroutine by **GOSUB 4000**



```
4000 REM HIT
4010 X=X-8
4020 IF X<256 THEN POKE V+2,X: POKE V+16,0
4030 IF X>255 THEN POKE V+2,X-255: POKEV+16,2
4040 POKE V+3,Y: POKE V+21,2
4050 FOR A=1 TO INVADERS
4060 IF X(A)<>XP OR Y(A)<>YM THEN GOTO 4080
4070 Y(A)=22: A=INVADERS
4080 NEXT A
4090 ALIENS=ALIENS-1: SC=SC+1: AHIT=0
4100 IF ALIENS=0 THEN WIN=1
4110 Y=YM: GOSUB 9000: PRINTSPC(XP)" "
4120 GRID%(XP,YM)=GRID%(XP,YM)-2
4130 PRINT"■";PRINTSPC(27)"■";SC
4140 YM=1
4150 RETURN
```

The routine above displays the explosion sprite on screen after a direct hit has been scored on an alien. The X and Y location registers and the two registers on the VIC chip are used to as they were in the last subroutine. In line 4030, though, sprite number two is enabled by **POKE**ing **V+16** with 2.

After a hit, line 4070 adjusts the alien's position so that it is off screen – by altering its Y coordinate to 22. Line 4090 alters the score, and the number of **ALIEN**s remaining, and then resets the **AHIT** flag. The alien is blanked out by calling the subroutine at line 9000 which positions the cursor over the alien and control **RETURN**s to the mains program.



Testing your program

You can test the moving and firing sections of the program at this point before going on to the final chapter of the book. Follow the instructions given in the boxes below. You should be able to move your spaceship and attack the oncoming invaders. As before, if you receive an error message, check back through your work with the listings given.

APPLE IIe

1. TYPE **RUN** AND HIT RETURN

IF EVERYTHING IS CORRECT, YOU SHOULD BE ABLE TO SHOOT THE ALIENS AS THEY ADVANCE DOWN THE SCREEN. THE PROGRAM WILL STOP IF YOU KILL THEM ALL, OR ONE OF THEM REACHES THE BOTTOM OF THE SCREEN. DELETE THE DUMMY LINE BEFORE PROCEEDING. IGNORE THE ERROR MESSAGE PRINTED AT THIS STAGE.

COMMODORE

1. TYPE **RUN** AND HIT RETURN

IF EVERYTHING IS CORRECT, YOU SHOULD BE ABLE TO SHOOT THE ALIENS AS THEY ADVANCE DOWN THE SCREEN. THE PROGRAM WILL STOP IF ONE OF THEM REACHES THE BOTTOM OF THE SCREEN. DELETE THE DUMMY LINE BEFORE PROCEEDING. IGNORE THE ERROR MESSAGE PRINTED AT THIS STAGE.

WINNING AND LOSING

The winning and losing section is all that remains to make the game complete. If the player succeeds in shooting all ten aliens, another wave of attack appears, beginning lower down the screen and giving the player less time to deal with the attack. At the end of this section there are some tips to make the program even more effective, and the program listing is given in full.

```
THE
INVADERS
HAVE
LANDED

HOWEVER
YOU DESTROYED  5
ALIEN CRAFT

DO YOU WISH TO FIGHT AGAIN ?
```

If the player is skilful enough to destroy all the aliens in a wave, then the **WIN** flag is set in the **HIT** subroutine, and the control program calls the **WIN** subroutine. If the player lets the aliens through, then the **LOSE** subroutine is called instead.

```
5000  REM       WIN
5010  TEXT : HOME
5020  VTAB 12: HTAB 12: PRINT "WELL DONE EARTHLING"
5030  HTAB 13: PRINT "THIS TIME YOU WIN"
5040  HTAB 14: PRINT "NOW PREPARE FOR "
5050  HTAB 14: PRINT "OUR NEXT ATTACK"
5060  FOR D = 1 TO 1500: NEXT D
5070 ALIEN = INVADERS:HEIGHT = HEIGHT + 2:WIN = 0
5080  FOR A = 1 TO INVADERS
5090 X(A) = A * 3:Y(A) = HEIGHT
5100  NEXT A
5110  RETURN
```

Line 5010 clears the screen and sets the cursor to the **HOME** position (the top left hand corner of the screen) and switches to **TEXT** mode. Lines 5020, 5030 and 5040 **PRINT** a "Well Done" message in the top half of the screen, telling the player that another wave of aliens is on its way. Notice that each message line is preceded by **HTAB** so that the complete message is arranged in the correct place in the center of the screen. A **FOR . . . NEXT** loop at line 5060 introduces a time delay so the player can read the message.

This kind of delay asks the computer to count from one to 1500 – while it is counting it cannot get on with the next part of the program. If the message was shorter, the computer could be asked to count to a smaller number (a shorter delay), or if the message was longer, the computer could be asked to count to a larger number (a longer delay).

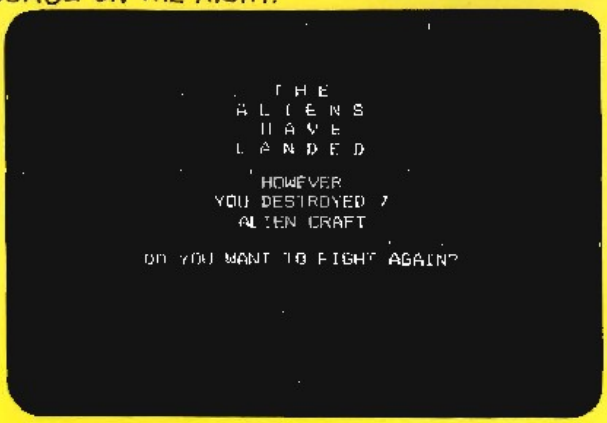
The number of **ALIEN**s is reset to the number of **INVADERS** in line 5070 and the **WIN** flag is switched "off" – i.e. becomes zero – ready for the next wave of aliens. The value of **HEIGHT** is increased by 2, making the next wave of aliens appear lower down the screen, and increasing the level of difficulty. The **FOR . . . NEXT** loop between line 5080 and line 5100 calculates the X and Y positions of each alien, by working out the X position from the alien number (**A**), and taking the Y position from the **HEIGHT** value. This is just the same as was done earlier in the **INITIALIZE** subroutine. This latest subroutine can be tested as before, by following the instructions in the box on the facing page.

```
6000  REM   LOSE
6010  TEXT : HOME
6020  VTAB 5: HTAB 15: PRINT "T H E"
6030  HTAB 12: PRINT "A L I E N S"
6040  HTAB 14: PRINT "H A V E "
6050  HTAB 12: PRINT "L A N D E D"
6060  VTAB 12: HTAB 14: PRINT "HOWEVER"
6070  HTAB 10: PRINT "YOU DESTROYED ";SC
6080  HTAB 12: PRINT "ALIEN CRAFT"
6090  VTAB 20: HTAB 4: PRINT "DO YOU WISH TO FIGHT
AGAIN?"
6100  POKE - 16368,0: GET A$
6110  IF A$ = "Y" OR A$ = "y" THEN  RUN
6120  RETURN
```

When the aliens land the **LOSE** subroutine (above) is called. The high resolution screen is replaced by the text screen for the message.

The screen is cleared and a message is **PRINT**ed on the **TEXT** screen, telling the player that the invaders have landed. Next, lines 6060 to 6080 tell the player his or her score. The semi colon (;) in line 6070 ensures that the score appears alongside the "YOU ARE DESTROYED" message. After a pause, the player is given the option of playing again. If the response is a Y, then the program **RUN**s again, otherwise the program **RETURN**s to the control program, and **END**s at line 30.

APPLE IIe *YOU CAN NOW* ***RUN*** *THE PROGRAM IN FULL. WHEN THE INVADERS ARE ALL SHOT DOWN, THE MESSAGE SHOWN LEFT IS DISPLAYED, AND THE GAME CONTINUES WITH THE ALIENS ADVANCING FROM A POSITION LOWER DOWN THE SCREEN. IF AN INVADER REACHES THE BOTTOM OF THE SCREEN, YOU HAVE LOST AND THE SCREEN WILL DISPLAY THE MESSAGE ON THE RIGHT.*



If the player is skilful enough to destroy all the aliens in a wave, then the **WIN** flag is set in the **HIT** subroutine, and the control program calls the **WIN** subroutine. If the player lets the aliens through, then the **LOSE** subroutine is called instead.

```
5000 REM WIN
5010 PRINT""
5020 PRINTSPC(10)"WELL DONE EARTHLING"
5030 PRINTSPC(11)"THIS TIME YOU WIN"
5040 PRINTSPC(12)"NOW PREPARE FOR"
5050 PRINTSPC(12)"OUR NEXT ATTACK"
5060 FOR D=1 TO 1500: NEXT D
5070 LET ALIENS=INVADERS: HEIGHT=HEIGHT+2: WIN=0
5080 FOR A=1 TO INVADERS
5090 LET X(A)=A*3: Y(A)= HEIGHT
5100 GRID%(X(A),Y(A))=2
5110 NEXT A
5120 RETURN
```

Lines 5010, 5020 and 5030 **PRINT** a "Well Done" message in the top half of the screen, telling the player that another wave of aliens is on its way. Notice that each message line is preceded by **SPC** so that the complete message is arranged in the correct place in the center of the screen. A **FOR . . . NEXT** loop at line 5060 introduces a time delay so the player can read the message. This kind of delay asks the computer to count from one to 1500 – while it is counting it cannot get on with the next part of the program. If the message was shorter, the computer could be asked to count to a smaller number (a shorter delay), or if the message was longer the computer could be asked to count to a larger number (a longer delay).

The number of **ALIEN**s is reset to the number of **INVADERS** in line 5070, and the **WIN** flag is switched "off" – i.e. becomes zero – ready for the next wave of aliens. The value of **HEIGHT** is increased by 2, making the next wave of aliens appear lower down the screen, and increasing the level of difficulty.

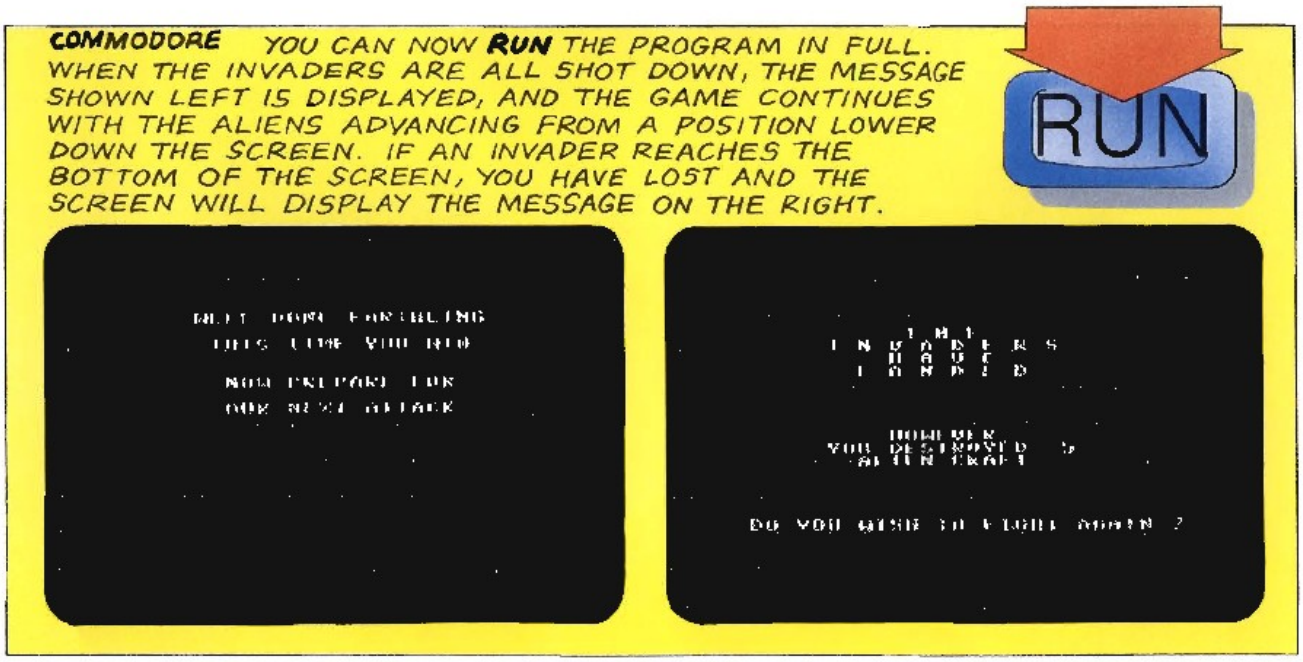
The **FOR . . . NEXT** loop between line 5080 and line 5110 repositions the X and Y positions of each alien, by working out the X position from the alien number (**A**), and taking the Y position from the **HEIGHT** value. This is just the same as was done earlier in the **INITIALIZE** subroutine. This latest subroutine can be tested as before, by following the instructions in the box on the facing page.

```
6000 REM LOSE
6010 PRINT"[CLR]"
6020 PRINTSPC(17)"[RVS]  [OFF]  [RVS]  [OFF]"
6030 PRINTSPC(12)"[RVS]  [OFF]  [RVS]  [OFF]  [RVS]  [OFF]"
6040 PRINTSPC(16)"[RVS]  [OFF]  [RVS]  [OFF]"
6050 PRINTSPC(14)"[RVS]  [OFF]  [RVS]  [OFF]  [RVS]  [OFF]"
6060 PRINTSPC(16)"[RVS]  [OFF]  [RVS]  [OFF]"
6070 PRINTSPC(12)"YOU DESTROYED ";SC
6080 PRINTSPC(14)"ALIEN CRAFT"
6090 PRINTSPC(7)"[RVS]  [OFF]  [RVS]  [OFF]  [RVS]  [OFF]"
     AGAIN";:INPUT A$
6100 IF LEFT$(A$,1)="Y" THEN RUN
6110 RETURN
```

The final section to be defined is the **LOSE** subroutine, starting at line 6000. The screen is cleared and a message is **PRINT**ed, telling the player that the invaders have landed. Next, lines 6060 to 6080 tell the player his or her score. The semi colon (;) at the end of line 6070 ensures that the score of the number of aliens killed (**SC**) appears alongside the "YOU DESTROYED" message.

The player is given the option of playing the game again after a pause. Anything the player types is given the label **A$**. If the response is any word which starts with a Y, line 6100 **RUN**s the program again, otherwise the program **RETURN**s to the control program, and stops at the **END** statement in line 90.

COMMODORE *YOU CAN NOW* **RUN** *THE PROGRAM IN FULL. WHEN THE INVADERS ARE ALL SHOT DOWN, THE MESSAGE SHOWN LEFT IS DISPLAYED, AND THE GAME CONTINUES WITH THE ALIENS ADVANCING FROM A POSITION LOWER DOWN THE SCREEN. IF AN INVADER REACHES THE BOTTOM OF THE SCREEN, YOU HAVE LOST AND THE SCREEN WILL DISPLAY THE MESSAGE ON THE RIGHT.*



Improve your program

As it stands, the Invaders game is fun and challenging to play. It is difficult to shoot down all the aliens and they appear to move as if they had a will of their own! However, it becomes much more like an arcade-style game if you add some simple sound effects for firing the missiles and alerting an alien attack. Given below are some hints on how you can add to the program to get a really professional-looking game that you and your friends can enjoy.

Adding sound

Add these lines to your program to give sound effects – a movement blip, an explosion noise, and a battle siren. Both the Apple and the Commodore generate sound effects when you **POKE** values into special memory locations. The Apple program has a special machine code routine which plays the sound effects, stored as **DATA** in line 9070. Every time a sound effect is needed, a value is **POKE**d into memory locations 776 and 777 to set pitch and note length before the machine code routine is called. The Commodore **POKE**s values into registers in a special sound synthesizer chip to produce its sounds.

APPLE IIe

```
1105  POKE 776,136: POKE 777,30: CALL 778: REM  BLIP
4055  POKE 776,254: POKE 777,100: CALL 778
7095  FOR I = 1 TO 20: POKE 776,90: POKE 777,30: CALL
8135  FOR E = 776 TO 798: READ B: POKE E,B: NEXT
9070  DATA  255,255,173,48,192,136,208,5,206,9,3,240,9
,202,208,245,174,8,3,76,10,3,96
```

COMMODORE

```
1105 GOSUB 9100: REM BLIP
4045 GOSUB 9200: REM EXPLOSION NOISE
7045 GOSUB 9300: REM BATTLE SIREN
```

```
9000 PRINT"■";: FOR I=1 TO Y: PRINT"■";: NEXT: RETURN
9100 REM BLIP
9110 POKE54276,0:POKE54277,0:POKE54277,0
9120 POKE54296,15:POKE54277,128:POKE54276,33
9130 POKE54273,30:POKE54272,75
9140 RETURN
9200 REM EXPLOSION NOISE
9210 POKE54283,129:POKE54284,15:POKE54250,40
9220 FOR P=15 TO 0 STEP -1:POKE 54296,P:NEXT
9230 POKE54283,0
9240 RETURN
9300 REM BATTLE SIREN
9310 FOR S=1 TO 20
9320 POKE54276,0:POKE54277,0:POKE54272,0
9330 POKE54296,15:POKE54277,64:POKE54276,33
9340 POKE54273,50:POKE54272,37
9350 NEXT
9360 RETURN
```

Extra Graphics

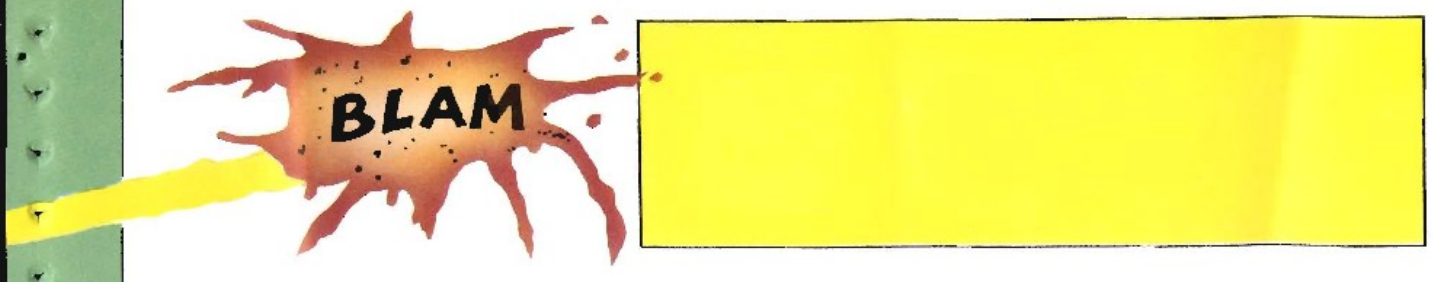
The graphics additions to the program are used after the aliens have landed. The Apple program uses the **FLASH** command in line 6015 to highlight the message printed in lines 6020 to 6050. **NORMAL** in line 6055 stops the following text from flashing. In the Commodore program the border is flashed by **POKE**ing the border color location with a range of color codes.

APPLE IIe

```
6015  FLASH
6055  NORMAL
```

COMMODORE

```
6085 FOR C=1 TO 15
6086 POKE53280,C
6087 FOR D=1 TO 300:NEXT
6089 NEXT
```



The complete listing

Now that you have the complete program keyed in, you should store it on tape for future use. The instructions for saving programs vary slightly between different machines, and you should consult your user's manual for detailed instructions. The full listing for the Space Invaders program is given below, for both machine versions. The additional material suggested on the previous page has been included.

APPLE IIe

```
5  REM   INVADERS
10  GOSUB 8000: REM  INITIALIZE
20  GOSUB 7000: REM   SCREEN
30  GOSUB 1000: REM   ALIEN
40  GOSUB 2000: REM   PLAYER
50  IF WIN = 1 OR LOST = 1 THEN  GOTO 70
60  GOTO 30
70  IF WIN = 1 THEN  GOSUB 5000: GOTO 20
80  GOSUB 6000: REM   LOSE
90  END
1000  REM   ALIEN
1010 R =  INT ( RND (1) * INVADERS) + 1: IF Y(R) = 22
THEN  RETURN
1020 GD%(X(R),Y(R)) = GD%(X(R),Y(R)) - 2
1070  HCOLOR= 0: DRAW 3 AT 8 * X(R),8 * Y(R)
1040 X(R) = X(R) +  INT ( RND (1) * 3) - 1
1050 Y(R) = Y(R) + 1
1060  IF X(R) > 33 THEN X(R) = 33
1070  IF X(R) < 1 THEN X(R) = 1
1080  IF Y(R) = 19 THEN LOST = 1
1090  HCOLOR= 5: DRAW 3 AT 8 * X(R),8 * Y(R)
1100 GD%(X(R),Y(R)) = GD%(X(R),Y(R)) + 2
1105  POKE 776,136: POKE 777,30: CALL 778: REM  BLIP
1110  RETURN
2000  REM     PLAYER
2010 I =  PEEK ( - 16384): IF I < 128 THEN 2070
2020 I$ =  CHR$ (I - 128)
2030  POKE  - 16368,0
2040  HCOLOR= 0: DRAW 2 AT 8 * XP,19 * 8
2050  IF (I$ = "Z" OR I$ = "z") AND XP > 1 THEN XP = X
P - 1
2060  IF (I$ = "X" OR I$ = "x") AND XP < 33 THEN XP =
XP + 1
2070  HCOLOR= 2: DRAW 2 AT 8 * XP,19 * 8
2080  IF I = 160 THEN  GOSUB 3000
2090  RETURN
3000  REM   FIRE
3010 XM = XP * 8
3020  FOR YM = 17 TO 0 STEP  - 1
3030  HCOLOR= 2: DRAW 1 AT XM,YM * 8
3040  IF GD%(XP,YM) > 1 THEN  GOSUB 4000
3050  HCOLOR= 0: DRAW 1 AT XM,YM * 8
3060  IF GD%(XP,YM) = 1 THEN  HCOLOR= 7: DRAW 4 AT XM,
YM * 8
3070  NEXT
3080  RETURN
4000  REM      HIT
4010  FOR A = 1 TO INVADERS
4020  IF X(A) <  > XP OR Y(A) <  > YM THEN  GOTO 4080
4030  HCOLOR= 0: DRAW 3 AT 8 * X(A),8 * Y(A)
4040  HCOLOR= 0: DRAW 1 AT 8 * X(A),8 * Y(A)
4050  HCOLOR= 7: DRAW 5 AT 8 * X(A),8 * Y(A)
4055  POKE 776,254: POKE 777,100: CALL 778
4060 GD%(X(A),Y(A)) = GD%(X(A),Y(A)) - 2
4070 Y(A) = 22:A = INVADERS
4080  NEXT A
4090 ALIEN = ALIEN - 1:SC = SC + 1:AHIT = 0
4100  IF ALIEN = 0 THEN WIN = 1
4110  VTAB 22: HTAB 27: PRINT SC
4120  HCOLOR= 0: DRAW 5 AT 8 * XP,8 * YM
4130 YM = 0
4140  RETURN
5000  REM     WIN
5010  TEXT : HOME
5020  VTAB 12: HTAB 12: PRINT "WELL DONE EARTHLING"
5030  HTAB 13: PRINT "THIS TIME YOU WIN"
5040  HTAB 14: PRINT "NOW PREPARE FOR "
5050  HTAB 14: PRINT "OUR NEXT ATTACK"
5060  FOR D = 1 TO 1500: NEXT D
5070 ALIEN = INVADERS:HEIGHT = HEIGHT + 2:WIN = 0
5080  FOR A = 1 TO INVADERS
5090 X(A) = A * 3:Y(A) = HEIGHT
5100  NEXT A
5110  RETURN
```

COMMODORE

```
5 REM INVADERS
10 GOSUB 8000: REM INITIALIZE
20 GOSUB 7000: REM SCREEN
30 GOSUB 1000: REM ALIEN
40 GOSUB 2000: REM PLAYER
50 IF WIN=1 OR LOST=1 THEN GOTO 70
60 GOTO 30
70 IF WIN=1 THEN GOSUB 5000: GOTO 20
80 GOSUB 6000: REM LOSE
90 END
1000 REM ALIEN
1010 R=INT(RND(1)*INVADERS)+1: IF Y(R)=22 THEN RETURN
1020 Y=Y(R): X=X(R): GOSUB 9000: PRINTSPC(X)" ";
1030 GRID%(X,Y)=GRID%(X,Y)-2
1040 IF GRID%(X(R),Y(R))=1 THEN GOSUB 9100:
     PRINTSPC(X)"■.";
1050 X(R)=X(R)+INT(RND(1)*3)-1
1060 Y(R)=Y(R)+1
1070 IF X(R)>36 THEN X(R)=36
1080 IF X(R)<1 THEN X(R)=1
1090 IF Y(R)=21 THEN LOST=1
1100 Y=Y(R): X=X(R): GOSUB 9000:PRINTSPC(X)"■■";
1105 GOSUB 9100: REM BLIP
1110 GRID%(X,Y)=GRID%(X,Y)+2
1120 RETURN
2000 REM PLAYER
2010 GET L$
2020 IF L$=" " THEN GOSUB 3000: REM FIRE
2030 IF L$="Z" AND XP>1 THEN XP=XP-1
2040 IF L$="X" AND XP<37 THEN XP=XP+1
2050 Y=22: GOSUB 9000: PRINTSPC(XP-1)"■■ A ";
2060 RETURN
3000 REM FIRE
3010 X=(XP*8)+24
3020 FOR YM=21 TO 1 STEP -1
3030 Y=(YM*8)+50
3040 IF X<256 THEN POKE V,X: POKE V+16,0
3050 IF X>255 THEN POKE V,X-255: POKE V+16,1
3060 POKE V+1,Y: POKE V+21,1
3070 IF GRID%(XP,YM)>1 THEN GOSUB 4000
3080 NEXT
3090 POKE V+21,0
3100 RETURN
4000 REM HIT
4010 X=X-8
4020 IF X<256 THEN POKE V+2,X: POKE V+16,0
4030 IF X>255 THEN POKE V+2,X-255: POKE V+16,2
4040 POKE V+3,Y: POKE V+21,2
4045 GOSUB 9200: REM EXPLOSION NOISE
4050 FOR A=1 TO INVADERS
4060 IF X(A)<>XP OR Y(A)<>YM THEN GOTO 4080
4070 Y(A)=22: A=INVADERS
4080 NEXT A
4090 ALIENS=ALIENS-1: SC=SC+1: AHIT=0
4100 IF ALIENS=0 THEN WIN=1
4110 Y=YM: GOSUB 9000: PRINTSPC(XP)" "
4120 GRID%(XP,YM)=GRID%(XP,YM)-2
4130 PRINT"■■":PRINTSPC(27)"■■";SC
4140 YM=1
4150 RETURN
5000 REM WIN
5010 PRINT"■■"
5020 PRINTSPC(10)"■■■■■■■WELL DONE EARTHLING"
5030 PRINTSPC(11)"■■THIS TIME YOU WIN"
5040 PRINTSPC(12)"■■■■NOW PREPARE FOR"
5050 PRINTSPC(12)"■■■OUR NEXT ATTACK"
5060 FOR D=1 TO 1500: NEXT D
5070 LET ALIENS=INVADERS: HEIGHT=HEIGHT+2: WIN=0
5080 FOR A=1 TO INVADERS
5090 LET X(A)=A*3: Y(A)=HEIGHT
5100 GRID%(X(A),Y(A))=2
5110 NEXT A
5120 RETURN
```

COMMODORE continued

```
6000 REM LOSE
6010 PRINT"■"
6020 PRINTSPC(17)"■■■ H E "
6030 PRINTSPC(15)"I N V A D E R S"
6040 PRINTSPC(16)"H A V E "
6050 PRINTSPC(14)"L A N D E D"
6060 PRINTSPC(16)"■■■■HOWEVER,"
6070 PRINTSPC(12)"YOU DESTROYED ";SC
6080 PRINTSPC(14)"ALIEN CRAFT"
6085 FOR I=1 TO 15
6086 POKE53280,I
6087 FOR D=1 TO 50:NEXT
6089 NEXT
6090 PRINTSPC(2)"■■■■■DO YOU WISH TO FIGHT
     AGAIN";:INPUT A$
6100 IF LEFT$(A$,1)="Y" THEN RUN
6110 RETURN
7000 REM SCREEN
7010 POKE V+21,0: PRINT"■"
7020 FOR S=1 TO 40
7030 X=INT(RND(1)*40): Y=INT(RND(2)*21): PRINT"■";
     GOSUB9000: PRINTTAB(X)"■";
7040 GRID%(X,Y)=1
7050 NEXT S
7060 PRINT"■";:PRINTSPC(10)"ALIENS DESTROYED=";SC
7065 GOSUB 9300: REM BATTLE SIREN
7070 RETURN
8000 REM INITIALIZE
8010 POKE 53280,0: POKE 53281,0: POKE 650,255:
     PRINT"■"
8020 PRINTSPC(6)"■YOU ARE A LONE SPACE PILOT"
8030 PRINTSPC(4)"PROTECTING THE PLANET
     EARTH. IN"
8040 PRINTSPC(4)"A FEW MOMENTS YOU WILL BE
     UNDER"
8050 PRINTSPC(4)"ATTACK BY ALIENS FROM THE
     PLANET"
8060 PRINTSPC(4)"VARGON"
8070 PRINTSPC(4)"YOUR MISSION IS TO PREVENT
     ANY"
8080 PRINTSPC(4)"VARGONIAN SHIP FROM
     LANDING."
8090 PRINTSPC(6)"■■■'Z' MOVE S STARFIGHTER
     LEFT"
8100 PRINTSPC(6)"'X' MOVES STARFIGHTER RIGHT"
8110 PRINTSPC(7)"■PRESS SPACE TO FIRE LASER"
8120 PRINTSPC(5)"■■■PRESS ANY KEY TO ENGAGE
     ENEMY"
8130 LOST=0: WIN=0: SC=0: AHIT=0: HITSTAR=0: XP=20:
     HEIGHT=1: INVADERS=10
8140 ALIENS=INVADERS
8150 DIM X(INVADERS): DIM Y(INVADERS):
     DIM GRID%(40,21)
8160 FOR A=1 TO INVADERS
8170 X(A)=A*4-2: Y(A)=HEIGHT
8180 GRID%(X(A),Y(A))=2
8190 NEXT A
8200 GOSUB 9000
8210 A$=""
8220 GET A$: IF A$="" THEN 8220
8230 RETURN
8500 REM SPRITE DEFINITION
8510 FOR N=0 TO 62: READ D: POKE 832+N,D: NEXT:
     REM MISSILE
8520 FOR N=0 TO 62: READ D: POKE 896+N,D: NEXT:
     REM EXPLOSION
8530 V=53248
8540 POKE 2040,13
8550 POKE 2041,14
8560 POKE V+39,2
8570 POKE V+40,7
8580 DATA 8,0,0,28,0,0,28,0,0,62,0,0,93,0,0,28,0,0,
     28,0,0,62,0,0,127,0,0,127,0,0
8590 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
     0,0,0,0,0,0,0,0,0,0,0,0,0,0
8600 DATA 0,0,0,176,0,17,2,0,64,1,0,128,16,165,8,
     129,0,129,33,0,132,16,165,72
8610 DATA 12,90,48,18,165,72,0,92,0,0,92,0,72,165,18,
     48,90,12
8620 DATA 72,165,18,132,0,33,129,0,129,8,165,16,
     128,0,1,64,0,2,17,0,136
8630 RETURN
9000 PRINT"■": FOR I=1 TO X: PRINT"■";: NEXT: RETURN
9100 REM BLIP
9110 POKE54276,0:POKE54277,0:POKE54272,0
9120 POKE54296,15:POKE54277,128:POKE54276,33
9130 POKE54273,30:POKE54272,75
9140 RETURN
9200 REM EXPLOSION NOISE
9210 POKE54283,129:POKE54284,15:POKE54280,40
9220 FOR P=15 TO 0 STEP -1:POKE 54296,P:NEXT
9230 POKE54283,0
9240 RETURN
9300 REM BATTLE SIREN
9310 FOR S=1 TO 20
9320 POKE54276,0:POKE54277,0:POKE54272,0
9330 POKE54296,15:POKE54277,64:POKE54276,33
9340 POKE54273,50:POKE54272,37
9350 NEXT
9360 RETURN
```

APPLE IIe continued

```
6000  REM   LOSE
6010  TEXT : HOME
6015  FLASH
6020  VTAB 5: HTAB 15: PRINT "T H E"
6030  HTAB 12: PRINT "A L I E N S"
6040  HTAB 14: PRINT "H A V E '
6050  HTAB 12: PRINT "L A N D E D"
6055  NORMAL
6060  VTAB 12: HTAB 14: PRINT 'HOWEVER'
6070  HTAB 10: PRINT "YOU DESTROYED ";SC
6080  HTAB 12: PRINT "ALIEN CRAFT"
6090  VTAB 16: HTAB 4: PRINT "DO YOU WISH TO FIGHT
AGAIN?"
6100  POKE  - 16368,0: GET A$
6110  IF A$ = "Y" OR A$ = "y" THEN  RUN
6120  RETURN
7000  REM  SCREEN
7010  HGR : HOME
7020  POKE  - 16301,0: POKE 232,252: POKE 233,29: SCAL
E= 1
7030  FOR S = 1 TO 20
7040 DF =INT ( RND (1) * 35):GH =INT ( RND (1) * 1 9)
7050 GD%(DF,GH) = 1
7060  HCOLOR= 7: DRAW 4 AT 8 * DF,8 * GH
7070  NEXT S
7080  VTAB 22: HTAB 10: PRINT "ALIENS DESTROYED=";SC
7090  HCOLOR= 2: DRAW 2 AT 8 * XP,8 * 19
7095  FOR I = 1 TO 20: POKE 776,90: POKE 777,30: CALL
778: NEXT
7100  RETURN
8000  REM     INITIALIZE
8010  TEXT : HOME
8020  VTAB 2: HTAB 7: PRINT "YOU ARE A LONE SPACE
PILOT"
8030  HTAB 5: PRINT "PROTECTING THE PLANET EARTH. IN"
8040  HTAB 5: PRINT "A FEW MOMENTS YOU WILL BE UNDER"
8050  HTAB 5: PRINT "ATTACK BY ALIENS FROM THE PLANET"
8060  HTAB 16: PRINT "VARGON."
8070  HTAB 5: PRINT "YOUR MISSION IS TO PREVENT ANY"
8080  HTAB 5: PRINT "VARGONIAN SHIP FROM LANDING."
8090  VTAB 10: HTAB 7: PRINT "'Z' MOVES STARFIGHTER
LEFT"
8100  VTAB 12: HTAB 7: PRINT "'X' MOVES STARFIGHTER
RIGHT"
8110  VTAB 14: HTAB 7: PRINT "PRESS SPACE BAR TO FIRE
MISSILE"
8120  VTAB 16: HTAB 5: PRINT "PRESS ANY KEY TO ENGAGE
ENEMY"
8130  GOSUB 9000
8135  FOR E = 776 TO 798: READ B: POKE E,B: NEXT
8140 LOST = 0:WIN = 0:SC = 0:AHIT = 0:HITSTAR = 0:XP =
 16:HEIGHT = 0:INVADERS = 10:ALIEN = INVADERS
8150 ALIENS = INVADERS
8160  DIM X(INVADERS): DIM Y(INVADERS): DIM GD%(35,19)
8170  FOR A = 1 TO INVADERS
8180 X(A) = A * 3:Y(A) = HEIGHT
8190  NEXT A
8200  GET A$
8210  RETURN
9000  FOR X = 7676 TO 7904
9010  READ A
9020  POKE X,A
9030  NEXT X
9040  DATA  5,0,12,0,25,0,53,0,80,0,86,0,73,169,63,46,
53,63,46,53,63,22,45,5,00,73,45,169,58,223,27,55,109,7
3,53,255,219,55,109,73,53,63,63,63,55,54,37,108,73,53,
55,45,00,41,77,9,245,59,63,159,45,45,45,255,219,55,45,
45,45
9050  DATA  245,59,63,159,46,108,73,53,63,00,73,145,58
,46,5,00,13,77,13,182,58,223,36,220,147,46,61,22,109,1
13,45,109,57,4,173,36,8,193,173,32,36,223,255,27,255,5
1,150,173,191,82,118,182,41,69,41,5,193,193,1,00
9060  RETURN
9070  DATA  255,255,173,48,192,136,208,5,206,9,3,240,9
,202,208,245,174,8,3,76,10,3,96
```

Glossary

Array An array is a set of data, held together and identified by one variable name (see also the entry for *variable*). One way of imagining an array is as a series of boxes within the computer's memory, with each separate piece of data held in a separate box.

DELAY Delays are sometimes included in computer programs when it is necessary to slow the computer down. They are usually part of a **FOR . . . NEXT** loop (see below) and look like this in a program:
FOR DE = 1 TO 1000: NEXT DE.
This would cause the computer to count to 1000 before going on to the next stage of the program.

DIM The BASIC instruction for opening an array. It is followed by a number in brackets which tells the computer how big the array should be.

DRAW Is an APPLE BASIC instruction to **DRAW** a shape on the screen. It takes the form **DRAW N AT X,Y** where **N** is the shape table number and **X** and **Y** are the screen co ordinates where the shape is to be drawn.

flag A **flag** is an operator within a program that can be "set" to either "on" or "off", depending on certain conditions. These are often used in games to determine whether or not a game is won or lost – if the game is won, then a "win flag" can be set from "off", to "on" and the appropriate action is then taken. Flags are given the values of either **0** or **1**, corresponding to "off" and "on" respectively.

GOSUB GOSUB XXXX sends control of the program to a subroutine starting at line XXXX. The search for line XXXX starts at line 0 – so the program will run faster if subroutines that are called most often are placed near the start of the listing.

GOTO This instruction tells the computer to go to the specified line, missing out any lines in-between. It is used with **IF . . . THEN** (see below). Be careful when using **GOTO**s, as it's easy to have the program jumping backwards and forwards so much that it is impossible to read.

IF . . . THEN This is used as a way of telling the computer to do something only when certain conditions are true. This instruction often looks something like this: **IF** score=10 **THEN PRINT** "WELL DONE, YOU'VE WON!!!"

INPUT This instruction allows the computer to be given information while a program is running. When the computer comes to an **INPUT** instruction it prints a question mark (or, for some computers, a different symbol) to prompt the user, and waits for the input to be given

INT **INT** is short for integer, and instructs the computer to make a whole number of a figure with decimal places in it. It is often used in conjunction with the **RND** command which instructs the computer to generate a random number (see below).

LET This is one way of giving the computer information. In some programs there may be statements such as: **X=10**
This simply means that the number ten is stored under the label X. It is often clearer to write:
LET X=10
The **LET** statement also gives rise to something that at first sight seems illogical, if not impossible. In many programs you will see things like:
LET X=X+1
Of course, in mathematical terms X can't equal X+1. All this type of statement means is "increase the value of whatever is stored in X by one."

LIST This makes the computer display whatever program is has in its memory. You can **LIST** single lines, or parts of a program by following the **LIST** command with the appropriate line numbers.

PEEK Is an instruction to look at the number which is contained within a specific memory location given in brackets. For example **PEEK** (12345) would look into memory location 12345.

POKE This instructs the computer to store a piece of information at a particular memory location. For example, the instruction: **POKE a,b** tells the computer to place the information **b** in memory location **a**.

RETURN This is the signal to end a subroutine. **RETURN** causes control of the program to go back to the statement following the most recently executed **GOSUB**.

RND This instruction makes the computer generate a random number. The precise instruction varies between different models of computer. Both the Apple IIe and the Commodore 64 generate random decimal numbers between 0 and 1. To make whole numbers between a desired range this is multiplied by a suitable figure and made a whole number using **INT**.

SPRITE In COMMODORE BASIC a sprite is a high resolution programmable object that can be made into any shape and moved about the screen using BASIC instructions. They are extremely useful for creating smooth animation - you can tell your sprite to move behind or in front of anything else on the screen. Even collisions between other sprites can be detected.

SHAPE TABLE In APPLE BASIC a shape table is a sequence of numbers that is stored in the computer's memory. These numbers can be used by a short routine that creates graphic figures on the screen by interpreting the numbers as "move only" or "plot and move" instructions. You can draw the shape anywhere on the screen using the DRAW instruction.

Index

Design
Cooper · West

Editor
David Rosam

Program editors
Steve Rodgers

Illustrators
Gerard Brown
Andrew Farmer

PRINTED IN BELGIUM BY
proost
INTERNATIONAL BOOK PRODUCTION

WRITE YOUR OWN PROGRAM

THIS NEW SERIES INTRODUCES THE ART OF PROGRAMMING YOUR COMPUTER. EACH BOOK SHOWS HOW TO STRUCTURE A PROGRAM INTO ROUTINES, AND AT THE SAME TIME EXPLAINS AND ANALYZES WHAT EXACTLY YOU ARE ASKING THE COMPUTER TO DO AND WHY.

AND THERE'S FUN TOO! EACH BOOK CONTAINS ONE OR MORE EXCITING AND ORIGINAL COMPUTER GAMES. THESE CAN BE ADAPTED AND EXTENDED TO DO BIGGER AND BETTER THINGS.

THE BOOKS ARE SPECIFIC TO THE COMMODORE 64 AND APPLE IIe.

TITLES IN THE SERIES

BEGINNING BASIC - SPACE JOURNEY
GRAPHICS - HANGMAN
MOVING GRAPHICS - ALIEN INVADERS
CREATING A DATABASE - ADVENTURE GAME

ISBN 0-531-03491-7

A GLOUCESTER PRESS LIBRARY EDITION