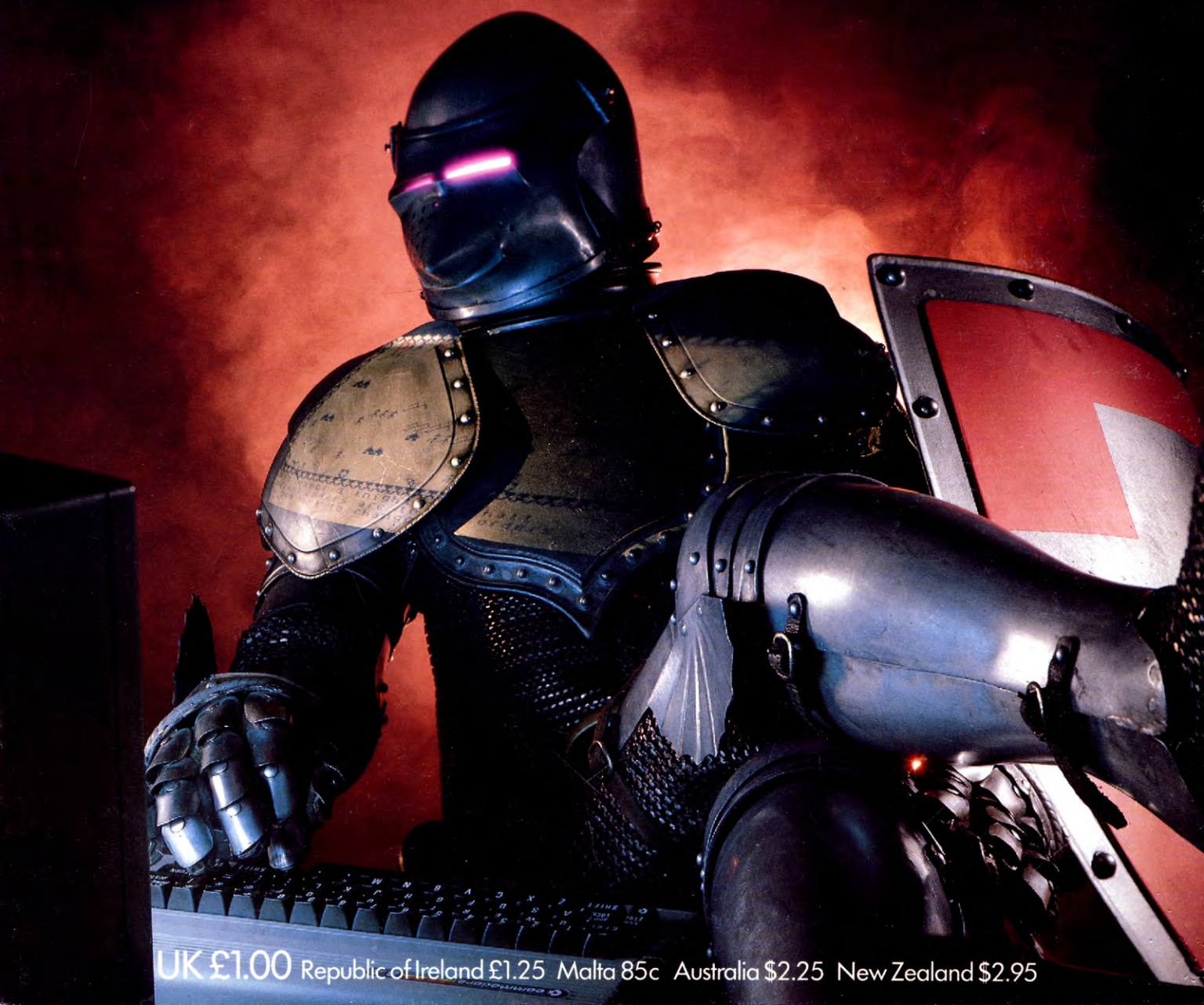
A MARSHALL CAVENDISH 40 COMPUTER COURSE IN WEEKLY PARTS

INPUT

LEARN PROGRAMMING - FOR FUN AND THE FUTURE



UK £1.00 Republic of Ireland £1.25 Malta 85c Australia $2.25 New Zealand $2.95

WARGAMING: FIRST STEPS

Designing a computer wargame is a fascinating project. Find out what's involved and then mobilize your micro when you start to enter INPUT's own tactical land battle game.

With the advent of computers, wargames have truly come of age. Despite wargames being around for thousands of years, playing them has been, until recently, a cumbersome process involving acres of floorspace and whole phalanxes of small models. A computer consigns all that to the dustbin, brings in a screen-based map, and may even be your opponent.

Wargames have, of course, a serious and deadly purpose, teaching the military how to fight wars more successfully. And though any wargame has, as its main aim, the killing of an enemy, it is certainly true that there is a great deal of interest to be had from creating and playing computer wargames. This interest, though, is more akin to that experienced in playing chess than the zap-pow excitement of arcade-type killing games.

The screen display of a wargame shows a map of the battle or war, usually with the positions of both sides shown. However, a computer game can have a greater degree of realism than traditional types because it is possible, if you wish, to have the opponent's units deployed on the board, but not displayed until their position has been discovered. Computer wargames may be for two players, or like the one you'll see developed in *INPUT*, a game in which one side is controlled by the computer.

The game the generals play tries, of course, to simulate every aspect of a real war as closely as possible. In a home computer game you won't be able to reproduce everything exactly. Nor would you want to—the result of including everything would be a game too horribly realistic for your players to find entertaining enough to play.

DESIGNING A WARGAME

War is about moving 'combat units' (which are usually human beings, but may be tanks or field guns) until they reach enemy units, when they attempt to compel each other to submit, almost always by fighting. The basic ingredients of a wargame, then, are two armies, their movements, and combat between the opposed combat units.

Starting with this as a base, you can consider the broad outline of the design of your wargame. Is the game to take place on land, sea or air? Is it to be a large-scale strategic game with many armies, a smaller tactical game between two armies on a single battlefield, or a skirmish between individual combatants?

Historical period will have a great influence over the type of game—what kind of technology (if any) is available to the combatants, and who are the combatants going to be.

You may wish to try to relive some historical battle, perhaps trying to force Napoleon's armies through to Moscow, or you may wish to invent your own conflict. At this stage, you can let your imagination run riot.

The next stage is to think of the 'rules' of your wargame, making decisions on all the details you want to include in your game, and under what circumstances they will help or hinder each army. For example, you may want your game to include a consideration of armour: wearing armour may protect a soldier during fighting, but it will slow him down as he moves to fight (or retreats after fighting).

It will be tempting to include as many of these components as you can think of, to make your wargame more realistic. But your computer's memory will limit the number of different points of detail you can include, and how complicated you can make your rules.

It's best to stick to the most important components:

- Geographical information on where the troops are and what the terrain is like—how it affects movement, and what cover it provides.
- Troops: how many are there; what weapons and armour do they have; how fast can they move?
- Movement: consider rules about how far units can move, and how far terrain affects movement.
- Orders: a mechanism for giving orders, and perhaps some possibility that the troops will disobey.
- Computer choice: how 'intelligent' will the computer be?
- Combat: what type? Combat is usually divided into missile combat and hand-to-hand combat. Missiles can be anything from throwing axes to intercontinental rockets.

SETTING UP THE GAME

Once the type, period and components of the game are determined you need to consider how they will be represented on your micro. There are two aspects to consider—how the game will appear to the player, and how the game will appear to the computer.

In this series of five articles you will see what is involved as you build a small wargame called Cavendish Field. It is a tactical land battle, fought between two medieval armies. However, the game isn't set in any particular period.

Like most computer wargames, Cavendish Field displays a map showing the disposition of the two armies and the terrain. The players (in this case, you and the computer) act as the commanders of their own units, and must make decisions on strategy, and issue appropriate commands to their men.

Full instructions on how to play the game will be given when the program is completed in part four of this article, but generally, the player is given the option of issuing new orders to each unit, or leaving them as they are. The game proceeds by taking turns to organize the disposition of the troops, which may or may not result in conflict. The outcome of any conflict is determined by the relative type and strengths of the combatants—plus a certain amount of luck. Play continues until one player has reduced the other's forces to an untenable level.

This game will show the important graphics information—the map, with the armies—continuously, but reserve an area for displaying the temporary text instructions and prompts.

It is best to use user defined graphics for the map display. This way, it is simple to handle the two areas of the display by using text-handling instructions for both. In the case of the Dragon and Tandy programs, UDGs have to be set up by DRAWing on the high resolution screen—so this has been used throughout, and a short routine for printing on the high resolution screen has been added.

You need to know what the UDGs will represent. In this game there are four kinds of terrain—plains, villages, woods and hills.

- WARGAMES BEFORE THE COMPUTER
- COMPUTER WARGAMES
- DESIGN CONSIDERATIONS
- HISTORICAL PERIOD
- THE RULES' ROLE
- SCREEN DISPLAY
- SETTING UP THE UDGS
- CLEARING THE TEXT AREA

Microtip

Detailed planning of wargames

Having decided on the broader aims of your wargame, it's very important to get the details right. You should decide on all the factors which are relevant to combat in your game. For example, you might wish to distinguish between missile conflict (whether it's arrows or neutron bombs) and hand-to-hand skirmishing. In most cases, you need at least two different missile types, two different troop types and some representation of range. As a missile weapon will be less effective at longer ranges, it becomes important to know how fast (or how far) the two troop types can move.

This might bring in considerations of armour. Heavy armour slows troops down, but protects them better. Armour will affect not only missile combat, but hand-to-hand combat.

Combat might also be affected by terrain (fighting knee-deep in nettles is harder than fighting in the open), cover (it is harder to hit a target behind a castle wall, than one behind a bush), the number of troops fighting, and so on.

At this planning stage, it's very important to bear in mind that you will almost certainly not be able to include everything you want in your game, even if you have a very large amount of memory available, so you have to balance the features in your game against the amount of RAM you have.

Blank space can be used to represent plains, so no UDGs are needed. Two UDGs each are needed to give a fair representation of hills, villages, units and forest.

Each army has eight units: one leader with his knights, a second body of knights (sergeants), two units of men-at-arms, two units of archers, and two of peasants. The two bodies of knights need a UDG each, but as each of the pairs of units will be armed with the same weapon, the same graphic can be used twice.

This gives a total of nine UDGs, as follows: number one, village; number two, forest; numbers three and four, hills; number five, leader (represented by a flag); number six, knights (a mace); number seven, men-at-arms (a shield); number eight, archers (a bow); and number nine, peasants (a sword).

SETTING UP THE UDGs

This section sets up nine UDGs.

```
210 FOR k=1 TO 9
220 READ a$
230 FOR I=Ø TO 7
240 READ a
260 POKE USR a$+I,a
270 NEXT I
290 NEXT K
2570 DATA "a",16,16,60,126,255,189,231,231
2590 DATA "b",16,56,84,16,56,84,146,16
2610 DATA "c",8,20,34,65,6,8,16,224
2630 DATA "d",0,48,72,132,2,0,0,0
2650 DATA "e",128,240,255,252,143,128,128,
   128
2670 DATA "f",64,240,72,68,68,68,78,68
2690DATA"g",255,231,231,129,129,231,102,
   60
2710 DATA "h",249,70,38,25,9,5,3,1
2730 DATA "i",1,2,4,8,16,160,64,160
```

```
10 CLR
21 PRINT CHR$(142)
22 POKE52,56:POKE56,56:CLR
23 POKE 56334,PEEK(56334)AND254
24 POKE1,PEEK(1)AND251
25 FORI=ØTO583:POKEI+14336,
   PEEK(I+53248):NEXT I
26 POKE1,PEEK(1)OR4
27 POKE56334,PEEK(56334)OR1
28 POKE 53272,(PEEK(53272)
   AND240)+14
29 FORI=ØTO71:READP:POKE(14848+I),P:
   NEXTI
2640 DATA 16,16,60,126,255,189,231,231
2650 DATA 16,56,84,16,56,84,146,16
2660 DATA 8,20,34,65,6,8,16,224
2670 DATA 0,48,72,132,2,0,0,0
2680 DATA 128,240,255,252,143,128,128,128
2690 DATA 64,240,72,68,68,68,78,68
2700 DATA 255,231,231,129,129,231,102,60
2710 DATA 249,70,38,25,9,5,3,1
2720 DATA 1,2,4,8,16,160,64,160
```

```
210 VDU23,224,&1010;&7E3C;&BDFF;&E7E7;
220 VDU23,225,&1C08;&082A;&2A1C;&0849;
230 VDU23,226,&0000;&8870;&0106;
   &0000;
240 VDU23,227,&2418;&01C2;&0806;&00F0;
250 VDU23,228,&F080;&F8FF;&809F;&8080;
260 VDU23,229,&0201;&0804;&E010;
   &A060;
270 VDU23,230,&F040;&4448;&4642;&464F;
280 VDU23,231,&E7FF;&8181;&66E7;
   &183C;
290 VDU23,232,&827D;&2945;&0911;&0205;
300 VDU19,2,4,0,0,0,19,3,2,0,0,0
320 COLOUR 131:COLOUR 0:CLS
```

```
210 FOR K=1 TO 9
220 READ A$
230 UC$(K)=A$
290 NEXT K
2570 DATA BR2D2L2D5RU2R3D2R3U4L2
   UL2D2
2590 DATA BR2R3DRDLDL2ND4L3U2R3DL3
2610 DATA BD3E2R2UD2RFGLG3
2630 DATA BD3E2RF3
2650 DATA ND7FR2FD5UNR3L2
2670 DATA ND7FR4DNR2DNFL3UR2
2690 DATA R5ND6DL5D5R3DL
2710 DATA NR3DRF6U4LHE2DL
2730 DATA BR7G7E2UNF3H2DR
```

The Spectrum program and the Commodore program use the same DATA, but treat it in different ways. The Spectrum sets up the UDGs in characters 124 to 133, but in the Commodore program, a protected block of memory must be reserved in the main program, into which ROM characters can be copied. The new characters are stored from location 14848 onwards, as characters 68 to 77.

The Acorn program simply uses VDU statements to set up the UDGs. In this case, they are set up in characters 224 to 232.

The Dragon/Tandy program uses DRAW instructions elsewhere in the program to set up the characters. In each of the DATA lines there is a string of instructions from which the characters can be outlined.

There is a drawback with using UDGs like this on the Commodore and Acorn machines. On the 64, although the program itself is only 6K in length, when RUN it consumes a great deal of extra RAM, owing to the size of the arrays used. The arrays mean that only a little over 10K of memory remains for any elaborations you may with to add. The problem is even more acute on the Acorns. In order to display a background colour, a colour for terrain and two different colours for the armies, yet have a map large enough for an interesting battle, MODE 1 is needed. This needs 20K of memory by itself, so all REM statements, and unnecessary spaces have been omitted from the listing in order to make sure that the program will run.

SCREEN DISPLAY

These few lines are necessary to clear the area assigned for printing up your instructions:

```
2540 REM Clear text screen
2550 FOR k=17 TO 21: PRINT AT k,0;
   "□□□□□□□□□□□□□□□□
   □□□□□□□□□□□□□□□□
   □□□": NEXT k
2555 RETURN
```

```
405 PS$="■■■■■■■■■■■■
   ■■■■■■■■■■■■"
2540 REM CLEAR TEXT WINDOW
2545 PRINTPS$;
2550 FOR K=1 TO 5:PRINT "□□
   □□□□□□□□□□□□□□□□
   □□□□□□□□□□□□□□□□
   □□□□□□□":NEXT K
2555 PRINTPS$;
2560 RETURN
```

```
2540 DEF PROCclean
2550 FOR k=23TO30:PRINT TAB(0,k);
   SPC(39):NEXT
2560 ENDPROC
```

```
2540 REM CLEAR TEXT
   WINDOW
2550 PMODE0,4:PCLS0: PMODE3,1
2560 RETURN
```

The programs treat the screen display as two 'windows'—a text window, and a map window.

The text window has to be cleared and re-written frequently during the game, but the map window has a constant display with occasional small movements of the units it shows.

All four machines see the screen as just one continuous section—not the two that the program needs. So these routines are designed to clear just the area set aside for the text window.

The next part of this article on wargaming deals with the map, and moving the different units around.

WARGAMING: OF MAPS AND MEN

Plan the map for Cavendish Field. Dot the battlefield with hills, forests and villages. And make the two warring factions draw up at their starting positions

In part one of this series of articles about writing computer wargames you set up the symbols for each of the military units needed for Cavendish Field. These symbols will be displayed (and moved) on a map, to show you the progress of the game, and allow you to plan your strategy.

PROGRAMMING THE MAP

Cavendish Field has an array to represent the map. The map will be displayed continuously in the game, taking the largest part of the screen display.

As the map is displayed all the time, strictly speaking, the array is unnecessary—there will always be an area of memory holding the screen which contains all the map's information. But it is worthwhile setting up the array, too, despite the memory sacrifice, because it can be read and written to with ease, rather than trying to POKE the information directly into areas of the screen memory.

The array has as many elements as there are screen positions within the map area. The Spectrum map is 30 by 16; the Commodore's map is 38 by 17; the Acorn's is 38 by 20; and the Dragon/Tandy's is 30 by 16.

The Acorn program runs in MODE 1, which alone consumes around 20K of memory. This leaves little memory left over for the program and the large map array—a normal array needs about five bytes per element, and an integer array needs about four bytes per element.

There simply isn't enough room to use an array of DIMensions 30 by 16, as dictated by the program design, so you need to use a new type of array—a byte array. This is simply a block of bytes determined by a DIMension statement.

The byte array used in Cavendish Field is a single-dimensional array and will need 760 elements, saving roughly 2K of memory over an integer array. Unfortunately, you'll need some more program lines to read from and write to a byte array.

THE TROOPS

To keep track of where the units are on the map—so that they can be moved, and to check if there are any obstacles in the way—you need another array, the troop array. But this array will be used for much more than just holding the position of the troops. In fact, it can hold every piece of information about them.

The troop array in Cavendish Field also holds information needed for combat, morale, movement and so on—exactly what is held in the troop array in any other game based on this structure will depend on the nature of the game you are writing, and each of these areas will be covered fully as you progress through the program.

SETTING UP THE ARRAYS

The following routine DIMensions the map and troop arrays:

```
350 DIM m(16,30)
355 DIM T(16,9)
```

```
350 DIM M(38,16)
355 DIM T(15,8)
```

```
25 DEF FNmread(x,y) = ?(map% + (x*20) + y)
350 DIM map%□760
355 DIM T%(15,8)
430 DEF PROCmput(x,y,val)
440 ?(map% + x*20 + y) = val
450 ENDPROC
```

```
350 DIM M(16,30)
355 DIM T(16,9)
```

All the programs simply DIMension the map array and the troop array. The map array is just the size of the screen display (except on the Acorns which work differently, as described above). The troop array is DIMensioned to hold 9 elements of information about each of 16 troop units. Where the numbers used are 15 and 8, the zero element of the array is being used, too.

In addition, the Acorn program DEFines a PROCedure to put values in the troop array, and a FuNction to read information from the troop array.

FILLING THE MAP

The next step is to determine the terrain, and the starting position of each unit and to display them on the screen. You could choose to set up a fixed map—if you wanted to try to duplicate a famous battle, you might decide to take this course. However, in most cases you will want to have a variety of maps open to you. The simplest way is to use your machine's random number generator.

Determining the terrain could involve making a simple random plot of a number of the terrain symbols—forest, hills and village. The problem with a simple random choice is that hills and woods are generally not dotted all over the place, but tend to concentrate in clumps. The program needs to take account of this.

Another consideration when starting to set up a map for a wargame is the kind of terrain in which you would expect your kind of battle to take place. For example, as Cavendish Field is a medieval wargame you would expect it to take place in fairly open terrain. In this case, you wouldn't aim to place too many hills and forests on the map.

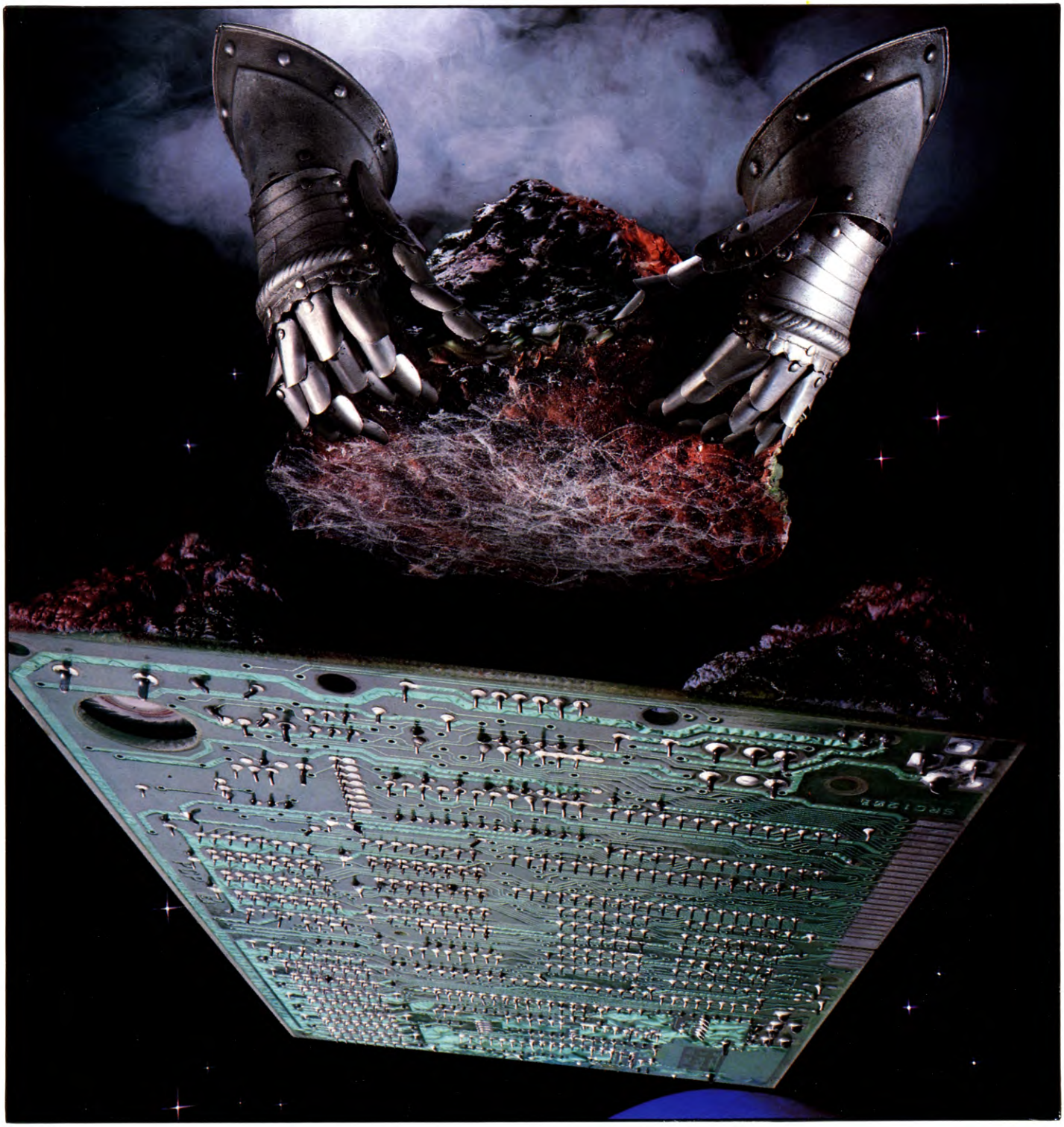
CHOOSING TERRAIN

The routine which chooses terrain is essentially random, but there is a degree of control over the selection. This ensures that the terrain is drawn realistically, with hills and woods clumped together.

```
20 DEF FN r(x) = INT (RND*x) + 1
800 REM Choose Terrain
810 LET R = FN r(50)
820 IF R > 5 THEN LET R = 0
830 IF R > 4 THEN LET R = 3: RETURN
840 IF R > 1 THEN LET R = 2
850 RETURN
```

```
800 REM CHOOSE TERRAIN
810 R = FNR(50)
820 IF R > 5 THEN R = 0
830 IF R > 4 THEN R = 3:RETURN
840 IF R > 1 THEN R = 2
850 RETURN
```

- THE MAP
- DIMENSIONING MAP AND TROOP ARRAYS
- CHOOSING REALISTIC TERRAIN
- DEPLOYING THE TROOPS
- USING STRINGS TO DISPLAY TROOP SYMBOLS
- DRAWING A BORDER
- FACTORS AFFECTING TROOP MOVEMENT



```
800 DEF PROCchter
810 R=RND(50)
820 IF R>5 THEN R=0
830 IF R>4 THEN R=3:ENDPROC
840 IF R>1 THEN R=2
850 ENDPROC
```

```
800 REM CHOOSE TERRAIN
810 R=RND(50)
820 IF R>5 THEN R=0
830 IF R>4 THEN R=3:RETURN
840 IF R>1 THEN R=2
850 RETURN
```

Integer random numbers are used throughout Cavendish Field. The Acorn, Dragon and Tandy machines can generate numbers in this form, but the Spectrum and Commodore cannot. Line 20 of the Spectrum program, and Line 195 of the Commodore program DEFines a FuNction to generate integer random numbers.

In each case, a random number, R, between 1 and 50 is generated. The Choose Terrain routine may change the value of R according to the value generated. If the generated value is greater than five, R is set to zero, the code for plains. If the generated value is five, R is set to three, the value representing hills; and if the value is two, three or four, R is set to two, the value representing forest. The remainder of cases are when R equals one—representing villages.

UNDER CONTROL

The Choose Terrain routine is called for each element along one dimension of the array, giving a random choice for the first terrain elements on the map. However, as was mentioned earlier, total randomness is not at all desirable. The next routine makes sure there is some pattern to the terrain upon which battle is to be fought.

```
370 LET i$="NWSE"
470 REM Create
480 FOR i=1 TO 16: GOSUB 800: LET
    m(i,1)=R: NEXT i
490 FOR i=1 TO 16
500 FOR j=2 TO 30
510 LET s=FN r(10)
520 IF s<8 THEN GOSUB 800
525 IF s>=8 THEN LET R=m(i,j-1)
530 LET m(i,j)=R
540 IF R=3 AND j<30 THEN LET
    m(i,j+1)=4
550 IF m(i,j)<>0 AND m(i,j)
    <>3 THEN PRINT AT i,j;CHR$
    (m(i,j)+143)
555 IF m(i,j)=3 AND j<>30
    THEN PRINT AT i,j;CHR$ 146;AT i,j+
    1;CHR$ 147
560 NEXT j
570 NEXT i
580 GOSUB 720
590 FOR i=1 TO 8
600 FOR j=1 TO 2: LET T(i,j)=2: LET
    T(i+8,j)=2: NEXT j
610 FOR j=3 TO 4: READ T(i,j): LET
    T(i+8,j)=T(i,j): NEXT j
620 READ mr
630 FOR j=0 TO 8 STEP 8
640 LET T(i+j,5)=mr+FN r(2)
650 LET T(i+j,6)=(FN r(100)*10)+10
660 LET T(i+j,7)=T(i+j,6)
670 NEXT j
680 LET T(i,8)=15
690 LET T(i+8,8)=1
700 NEXT i
710 RETURN
```

```
370 I$="NWSE"
470 REM CREATE
480 FOR I=0 TO 16:GOSUB800:M(0,I)=
    R:NEXT I
490 FOR I=0 TO 16
500 FOR J=1 TO 37
510 S=FNR(10)
520 IFS<8 THEN GOSUB 800
525 IF S>=8 THEN R=M(J-1,I)
530 M(J,I)=R
540 IF R=3 AND J<37 THEN
    M(J+1,I)=4
550 IF M(J,I)<>0 THENP=J:Q=I:
    GH=(M(J,I)+63):CL=0:
    GOSUB2600
555 IF M(J,I)=3 THENP=J+1:Q=I:GH=
    (M(J,I)+64):CL=0:
    GOSUB2600
560 NEXT J
570 NEXT I
580 GOSUB 720
590 FOR I=0 TO 7
600 FOR J=0 TO 1:T(I,J)=1:T(I+8,J)=
    1:NEXT J
610 FOR J=2 TO 3:READ T(I,J):T(I+8,J)=
    T(I,J):NEXTJ
620 READ MR
630 FOR J=0 TO 8 STEP 8
640 T(I+J,4)=MR + FNR(2)-1
650 T(I+J,5)=(FNR(100)*10)+10
660 T(I+J,6)=T(I+J,5)
670 NEXT J
680 T(I,8)=16
690 T(I+8,8)=0
700 NEXT I
710 RETURN
```

```
370 dir$ = "NWSE"
470 DEF PROCcr
480 FORi = ØTO19:PROCchter:
    PROCmput(Ø,i,R):NEXT
490 FORi = ØTO19
500 FORj = 1TO37
510 s = RND(1Ø)
520 IF s < 8 THEN PROCchter ELSE
    R = FNmread(j - 1,i)
530 PROCmput(j,i,R)
540 IFR = 3 AND j < 37 THEN
    PROCmput(j + 1,i,4)
550 IF FNmread(j,i) < > Ø AND FNmread
    (j,i) < > 3 THEN PRINT TAB(j + 1,i + 1);
    CHR$(FNmread(j,i) + 223) ELSE IF
    FNmread(j,i) < > Ø PRINT TAB(j + 1,i + 1);
    CHR$(227);CHR$(226)
560 NEXT:NEXT
580 PROCborder
590 FORi = ØTO7
600 FORj = ØTO1:T%(i,j) = 1:T%(i + 8,j) = 1:
    NEXT
610 FORj = 2TO3:READ T%(i,j):T%
    (i + 8,j) = T%(i,j):NEXT
620 READ mo
630 FORj = ØTO8STEP8
640 T%(i + j,4) = mo + RND(2) - 1:T%
    (i + j,5) = (RND(100)*10) + 10:T%
    (i + j,6) = T%(i + j,5)
670 NEXT
680 T%(i,8) = 20:T%(i + 8,8) = Ø
700 NEXT
710 ENDPROC
```

```
370 I$ = "NWSE"
470 REM CREATE
480 FOR I = 1 TO 16:GOSUB 800:
    M(I,1) = R:NEXT I
490 FOR I = 1 TO 16
500 FOR J = 2 TO 30
510 S = RND(10)
520 IF S < 8 THEN GOSUB 800
525 IF S > = 8 THEN R = M(I,J - 1)
530 M(I,J) = R
540 IF R = 3 AND J < 30 THEN M(I,J + 1) = 4
550 IF M(I,J) < > Ø  AND  M(I,J) < > 3
    THEN  LINE (J*8, I*8) -  (J*8 + 7, I*8 + 7),
    PRESET, BF: DRAW "BM" +  STR$
    (J*8) +  "," +  STR$ (I*8) +  UC$ (M(I,J))
555 IF M(I,J) = 3  AND  J < > 30  THEN
    DRAW "BM" +  STR$ (J*8) +  "," +
    STR$ (I*8) +  UC$(3) +  "BM" +  STR$
    ((J + 1)*8) +  "," +  STR$ (I*8) +  UC$(4)
560 NEXT J,I
580 GOSUB 720
590 FOR I = 1 TO 8
600 FOR J = 1 TO 2:T(I,J) = 2:T(I + 8,J) = 2:
NEXT J
610 FOR J = 3 TO 4:READ T(I,J):T(I + 8,J) = T
    (I,J):NEXT J
620 READ MR
630 FOR J = Ø TO 8 STEP 8
640 T(I + J,5) = MR + RND(2)
650 T(I + J,6) = (RND(100)*10) + 10
660 T(I + J,7) = T(I + J,6)
670 NEXT J
680 T(I,8) = 15
690 T(I + 8,8) = 1
700 NEXT I
710 RETURN
```

The routine generates a new random number, S, for each subsequent element of the array. The value of S may range from one to ten—chosen in Line 510. Line 520 ensures that, for seven tenths of the time, the new element in the array will have randomly generated terrain—if S < 8, then the program jumps to the Choose Terrain routine. For the other three tenths of the time, the element will have exactly the same terrain as that immediately to its left. This has the effect of creating blocks on the map. Lines 550 to 570 display the chosen terrain on the map.

DEPLOYING THE TROOPS

Troop positions are held in the troop array as a pair of coordinates—horizontal and vertical.

The starting positions of the opposing sides' units in Cavendish Field are at different ends of the map—the player's starting at the southern end (the very bottom of the screen display), and the computer's starting at the northern (the top of the display). The vertical coordinate, then, doesn't need to be chosen.

The horizontal coordinate, needs to be selected much like choosing the terrain. It needs to have an element of randomness, but some constraints have to be imposed. You must ensure that two or more units do not appear on the same square, for example.

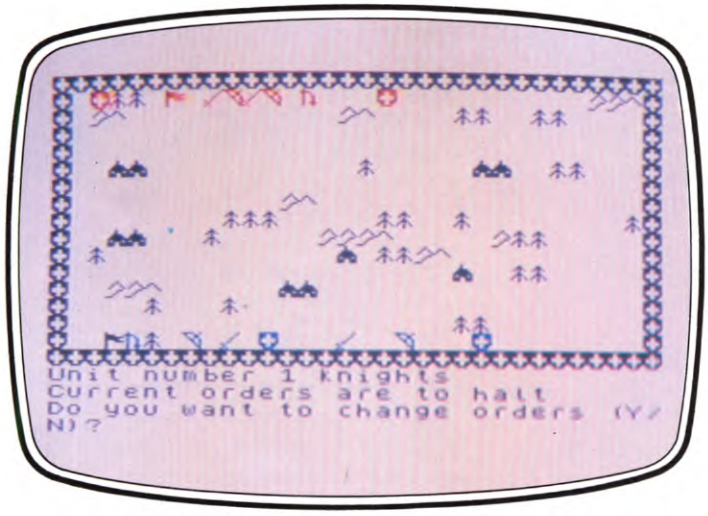
The following routine chooses the starting positions of each side. First it picks the troop units at random. Then it divides the map vertically into eight columns. Each of these represents the limits within which one of the units on each side will be placed. The actual print position is selected randomly within the limits of the column width.

```
860 REM Dispose Troops
870 INK 2
880 FOR m = 1 TO 2
890 LET s = 1: LET r = 1
900 FOR k = 1 TO 8
910 REM Dummy for Repeat loop
920 LET s = FN r(8*m)
930 IF T(s,9) < > Ø THEN GOTO 910
940 LET r = FN r(4) + r
950 LET r = r - INT (r/30)
960 LET T(s,9) = r
970 INK m
980 PRINT AT T(s,8),T(s,9);u$(s)
990 NEXT k
1000 NEXT m
1010 RETURN
```

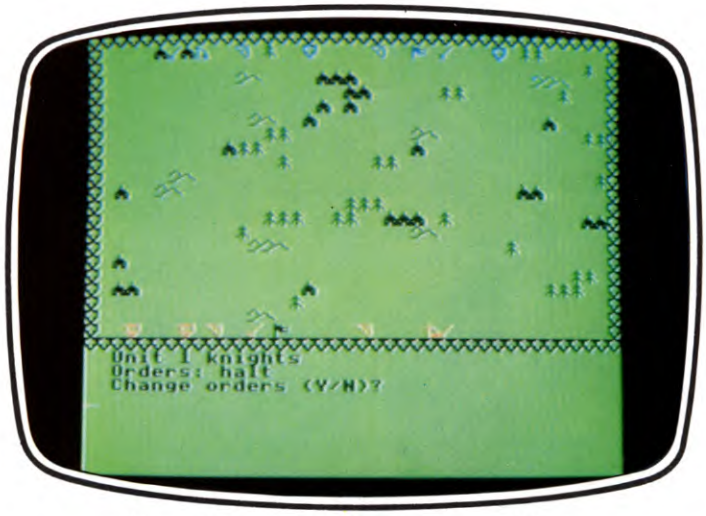
```
860 REM DISPOSE TROOPS
870 CL = 1
880 FOR M = 1 TO 2
890 S = Ø:R = Ø
900 FOR K = Ø TO 7
910 REM DUMMY FOR REPEAT LOOP
920 S = FNR(8*M) - 1
930 IF T(S,7) < > Ø THEN 910
940 R = FNR(6) + R
950 R = R - INT(R/37)
960 T(S,7) = R
970 IFM = 2 THEN CL = 9
980 P = T(S,7):Q = T(S,8):GH = VAL
    (MID$(U$,S + 1,1)) + 67:GOSUB2600
990 NEXT K
1000 NEXT M
1010 RETURN
```

```
860 DEF PROCds
870 COLOUR 2
880 FOR m = 1TO2
890 s = Ø:r = Ø
900 FORk = ØTO7
910 REPEAT
920 s = RND(8*m) - 1
930 UNTIL T%(s,7) = Ø
940 r = RND(6) + r
950 r = r □MOD 37
960 T%(s,7) = r
970 COLOUR m
980 PRINT TAB(T%(s,7) + 1,T%(s,8) + 1);
    MID$(unst$,(s□MOD 8) + 1,1)
990 NEXT:NEXT
1010 ENDPROC
```

```
860 REM DISPOSE TROOPS
870 COLOR 2
880 FOR M = 1 TO 2
890 S = 1:R = 1
900 FOR K = 1 TO 8
910 REM
920 S = RND(8*M)
930 IF T(S,9) < > Ø THEN 910
940 R = RND(4) + R
950 R = R - INT(R/30)
960 T(S,9) = R
970 COLOR M:IF M = 1 THEN COLOR 3
980 DRAW"BM" + STR$(T(S,9)*8) + "," +
    STR$(T(S,8)*8):UU = VAL(MID$(U$,S,1)):
```



Cavendish Field shown on the Spectrum



The battle zone on the Acorn machines

```
    A$=UC$(UU):GOSUB 3000
990 NEXT K
1000 NEXT M
1010 RETURN
```

Because each of the eight units is chosen at random, this helps each game to appear different, as a given unit doesn't always start in the same eighth of the map. As both armies are stored in the same array, the same routine can be used to choose positions for both armies—all you need is the FOR . . . NEXT loop between Lines 880 and 1000. The loop also ensures that the two armies appear in different colour.

ON TO THE BATTLEFIELD

Once the starting positions of the two armies have been determined, the units can be made ready for display on screen.

```
410 LET U$=CHR$148+CHR$149+CHR$
    150+CHR$150+CHR$151+CHR$ 151
    +CHR$152+CHR$152:LETU$=U$+U$
```

```
410 U$="12334455":U$=U$+U$
2600 REM DRAW
2610 POKE (1064+P+(Q*40)),GH
2620 POKE (55336+P+(Q*40)),CL
2630 RETURN
```

```
410 unst$=CHR$(228)+CHR$(230)+
    CHR$(231)+CHR$(231)+CHR$(232)+
    CHR$(232)+CHR$(229)+CHR$(229)
```

```
410 U$="65778899":U$=U$+U$
```

In Line 410 a string, U$ (or unst$, in the case of the Acorn) is set up. This holds the code for the symbol representing each unit.

The Commodore needs to POKE directly on to the screen, rather than use the PRINT AT or PRINT TAB that are available on the Spectrum or Acorn machines. You need two POKEs, in fact, one to the screen position and one to colour memory. Both POKEs require the same parameters, but of course, have different addresses.

Lines 2600 to 2630 of the Commodore program POKE the information to the correct addresses.

BORDERING ON HOSTILITY

The screen display can be made more attractive, and less misleading if a border is drawn round the map:

```
720 REM Decorative Border
730 FOR i=0 TO 16
740 PRINT AT i,0;CHR$ 150;AT i,31;CHR$
    150
750 NEXT i
760 FOR i=0 TO 31
770 PRINT AT 0,i;CHR$ 150;AT 16,i;CHR$
    150
780 NEXT i
790 RETURN
```

```
720 REM DECORATIVE BORDER
730 FOR I=0 TO 39
740 POKE (1024+I),70:POKE
    (1024+I+(40*18)),70
750 NEXT I
790 RETURN
```

```
720 DEF PROCborder
730 FORi=0TO39
740 PRINT TAB(i,0);CHR$(231);TAB(i,22);
    CHR$(231)
750 NEXT
760 FORi=0TO22:PRINT TAB(0,i);CHR$
    (231);TAB(39,i);CHR$(231):NEXT
790 ENDPROC
```

```
720 REM BORDER
730 LINE(0,128)-(255,135),PRESET,BF
740 LINE(248,0)-(255,191),PRESET,BF:LINE
    (4,4)-(252,132),PSET,B
790 RETURN
```

The routines simply draw a series of symbols (the same as used for one of the troop units) round the map.

MOBILIZING YOUR FORCES

Now that the map is set up, both sides will want to be able to move their units. Units are moved in response to orders—more about that later—but there are several things that must be checked before a unit can be moved around the battlefield:

- The program has to know what the maximum movement (number of squares) is for each unit. In Cavendish Field any troop unit's mobility depends solely on the weight of their armour, but in your games it could also depend on discipline, morale, exhaustion, and so on.
- Are there any advantages or bonuses to be had? (In this game, only the cavalry have bonuses, but you might give bonuses for charging, for travelling downhill, or having

an enthusiastic leader.)

- Are there any terrain hazards blocking the way? (It's up to you what effect terrain has, but here all terrain except plains reduces the distance a unit moves by one square.) Is another unit in the way?
- Has the edge of the map been reached?

Add this routine and the program will be able to test for these factors before moving any unit:

```
1160 REM Move unit
1170 LET ox=T(b,8): LET oy=T(b,9)
1175 LET z$="□"
1180 IF m(T(b,8),T(b,9))<>0 THEN LET
   z$=CHR$ (143+m(T(b,8),T(b,9)))
1190 LET D=5-T(b,4)
1200 IF b<3 OR b=9 OR b=10 THEN LET
   D=D+2
1210 LET v=T(b,2)-1
1215 LET up=0: LET al=v-2
1220 IF v/2-(INT (v/2))=0 THEN LET
   up=v-1: LET al=0
1230 REM Dummy for Repeat
1240 LET n1=T(b,9)+al: LET
   np=T(b,8)+up
1250 IF np<1 THEN LET np=1
1260 IF np>15 THEN LET np=15
1270 IF n1<1 THEN LET n1=1
1280 IF n1>30 THEN LET n1=30
1290 IF m(np,n1)>0 THEN LET D=D-1
1300 FOR k=1 TO 8
1310 IF (T(k,9)=n1 AND T(k,8)=np AND
   k<>b) THEN LET D=0
1315 IF (T(k+8,9)=n1 AND T(k+8,8)=np
   AND k+8<>b) THEN LET D=0
1320 NEXT k
1330 IF D>0 THEN LET T(b,9)=n1: LET
   T(b,8)=np: LET D=D-1
1340 IF D<>0 THEN GOTO 1230
1350 INK 0: PRINT AT ox,oy;z$
1360 INK cl: PRINT AT T(b,8),T(b,9);u$(i)
1370 RETURN
```

```
1160 REM MOVE UNIT
1170 OX=T(B,7):OY=T(B,8)
1175 GH=32
1180 IF M(T(B,7),T(B,8))<>0 THEN
   GH=M(T(B,7),T(B,8))+63
1190 D=4-T(B,3)
1200 IF B<2 OR B=8 OR B=9
   THEND=D+2
1210 V=T(B,1)-1
1215 UP=0:AL=V-2
1220 IF V/2 -INT(V/2)=0 THEN
   UP=V-1:AL=0
1230 REM DUMMY FOR REPEAT LOOP
1240 NL=T(B,7)+AL:NP=T(B,8)+UP
1250 IF NP<0 THEN NP=0
1260 IF NP>16 THEN NP=16
1270 IF NL<0 THEN NL=0
1280 IF NL>37 THEN NL=37
1290 IF M(NL,NP)>0 THEN D=D-1
1300 FOR K=0 TO 7
1310 IF(T(K,7)=NL AND T(K,8)=NP AND
   K<>B)THEN D=0
1315 IF(T(K+8,7)=NL AND T(K+8,8)=NP
   AND K+8<>B)THEN D=0
1320 NEXT K
1330 IF D>0 THEN T(B,7)=NL:
   T(B,8)=NP:D=D-1
1340 IF D<>0 THEN 1230
1350 CL=0:P=OX:Q=OY:
   GOSUB2600
1360 CL=CO:P=T(B,7):Q=T(B,8):
   GH=VAL(MID$(U$,B+1,1))+67:
   GOSUB2600
1370 RETURN
```

```
1160 DEF PROCmove(B%)
1170 ox=T%(B%,7)+1:oy=T%(B%,8)+1
1180 IF FNmread(T%(B%,7),T%(B%,8))<>0
   THEN oldter$=CHR$(223+ FNmread(T%
   (B%,7),T%(B%,8))) ELSE oldter$="□"
1190 D%=4-T%(B%,3)
1200 IF B%<2 THEN D%=D%+2
1210 dir=T%(B%,1)-1
1220 IF dir□DIV 2 =dir/2 THEN
   up=dir-1:al=0 ELSE up=0:al=dir-2
1230 REPEAT
1240 nal=T%(B%,7)+al:nup=T%
   (B%,8)+up
1250 IF nup<0 THEN nup=0
1260 IF nup>20 THEN nup=20
1270 IF nal<0 THEN nal=0
1280 IF nal>37 THEN nal=37
1290 IF FNmread(nal,nup)>0 THEN
   D%=D%-1
1300 FOR k=0 TO 7
1310 IF (T%(k,7) = nal□AND T%(k,8) =
   nup□AND k<>B%) OR (T%(k+8,7) =
   nal□AND T%(k+8,8) = nup□AND
   k+8<>B%) THEN D% = 0
1320 NEXT
1330 IF D%>0 THEN T%(B%,7)=nal:
   T%(B%,8) =nup:D%=D%-1
1340 UNTIL D%=0
1350 COLOUR0:PRINT TAB(ox,oy);oldter$
1360 COLOURcl:PRINT TAB(T%
   (B%,7)+1,T%(B%,8)+1);
   MID$(unst$,(i□
   MOD 8)+1,1)
1370 ENDPROC
```

```
1160 REM MOVE UNIT
1170 OX=T(B,8):OY=T(B,9)
1175 ZZ=0
1180 IF M(T(B,8),T(B,9))<>0 THEN
   ZZ=M(T(B,8),T(B,9))
1190 D=5-T(B,4)
1200 IF B<3 OR B=9 OR B=10 THEN
   D=D+2
1210 V=T(B,2)-1
1215 UP=0:AL=V-2
1220 IF (V/2)-INT(V/2)=0 THEN
   UP=V-1:AL=0
1230 REM
1240 NL=T(B,9)+AL:NP=T(B,8)+UP
1250 IF NP<1 THEN NP=1
1260 IF NP>15 THEN NP=15
1270 IF NL<1 THEN NL=1
1280 IF NL>30 THEN NL=30
1290 IF M(NP,NL)>0 THEN D=D-1
1300 FOR K=1 TO 8
1310 IF (T(K,9)=NL AND T(K,8)=NP AND
   K<>B) THEN D=0
1315 IF (T(K+8,9)=NL AND T(K+8,8)=NP
   AND K+8<>B) THEN D=0
1320 NEXT K
1330 IF D>0 THEN T(B,9)=NL:
   T(B,8)=NP:D=D-1
1340 IF D<>0 THEN 1230
1350 X9=OY*8:Y9=OX*8:IF ZZ<>0 THEN
   COLOR 4:LINE(X9,Y9)-(X9+7,Y9+7),
   PRESET,BF:DRAW"BM"+STR$(X9)+
   ","+STR$(Y9)+UC$(ZZ) ELSE LINE
   (X9,Y9)-(X9+7,Y9+7),PRESET,BF
1360 COLOR CL:DRAW"BM"+STR$
   (T(B,9)*8)+","+STR$(T(B,8)*8):UU=
   VAL(MID$(U$,I,1)):A$=UC$(UU):
   GOSUB 3000
1370 RETURN
```

The tests will increase, reduce, or completely prevent movement. The routine first 'remembers' the old position, and the terrain of that position. It then calculates the direction and the maximum movement.

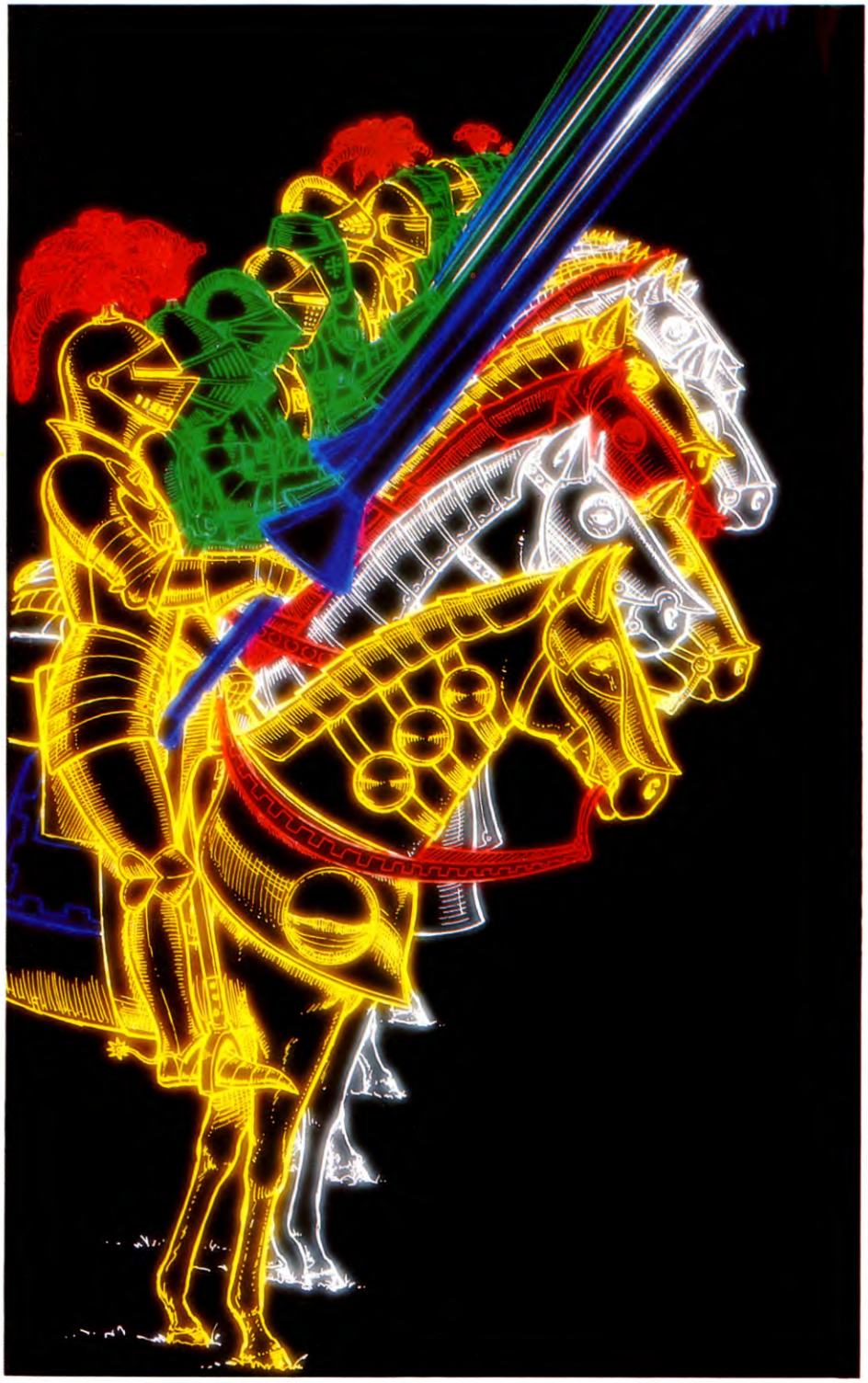
Lines 1230 to 1340 are a loop which tests each square along the unit's path to see if it is occupied by troops or terrain which would impede the unit's progress. According to what the loop finds, the distance the unit moves is adjusted.

The loop repeats until the distance variable is zero—the computer is back to the start square. A REPEAT . . . UNTIL loop is the most appropriate way to check the path, but it's only available on the Acorn, so on the other machines a dummy REPEAT . . . UNTIL loop has been set up. Several of these loops appear in the completed program, and are marked by a REM statement at the beginning of the loop.

Having decided on the range, the routine replaces the unit symbol with the original terrain. The unit is then displayed at its new position.

The next part of this article looks at how orders are issued to the units.

WARGAMING: THE ART OF COMMAND



Instil some discipline into the unruly rabble and stop them milling about the battlefield. Issue orders with these new routines and put your masterplan into action

Now you have the routines which will place your armies on the battlefield and you have added the movement routine, you will want to be able to issue commands to your army.

There are a number of factors connected with your troops which will affect the way they perform on the battlefield:

- Factor one, the unit's current order (what you told them to do last time)
- Factor two, the direction of current movement
- Factor three, weaponry
- Factor four, armour
- Factor five, initial strength
- Factor six, current strength
- Factor seven, morale or attitude
- Factor eight, position
- Factor nine, terrain

These factors correspond to the nine elements in the troop array that you set up last time. Position and terrain have already been filled, in connection with the movement routine, and in this article you will fill in the other elements.

At the start of the game initial values will have to be fed into the troop array. Weaponry and armour values will be fixed. Morale will be roughly the same each time, but may well vary slightly—peasants are probably not going to be very keen on fighting, whereas knights, having to keep up appearances, are never cowardly. Strength could vary quite considerably from game to game. Orders and direction could be set to arbitrary values at the start, but in Cavendish Field, everyone starts from Halt, meaning that direction is unimportant.

The following routines set up the troop array either by reading in appropriate data, or performing a calculation, or both. In fact, once you have the program working at the end of part four, you may wish to try experimenting with different ways of setting up these initial values.

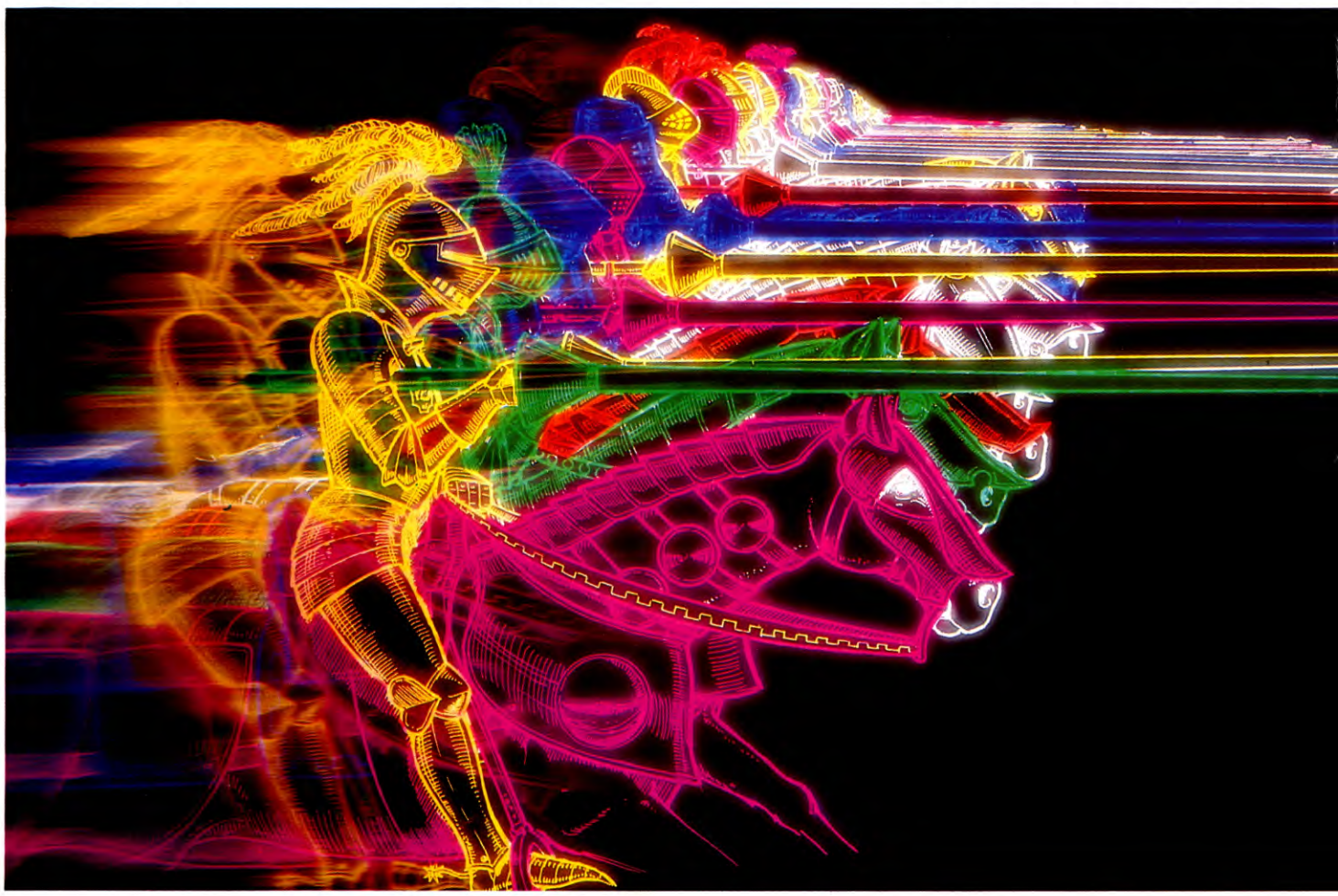
VISITING THE QUARTERMASTER

This routine initializes the unfilled array elements:

```
190 REM Initialization
200 LET vc=0: LET de=0
310 REM
320 PAPER 7
330 INK 0
340 CLS
360 DIM t$(8,12): DIM o$(5,12): DIM
    w$(5,9): DIM m$(5,12): DIM a$(4,12):
    DIMr$(4,12):DIMc(8)
380 FOR i=1 TO 5: READ o$(i),
    w$(i),m$(i): NEXT i
390 FOR i=1 TO 8: READ t$(i): NEXT i
400 FOR i=1 TO 4: READ a$(i),
    r$(i): NEXT i
410 LET u$=CHR$148+CHR$149+CHR$
    150+CHR$150+CHR$151+CHR$151+
    CHR$152+CHR$152:LET u$=u$+u$
415 LET x$="NnYy"
420 RETURN
2760 DATA "fire","none","cowardly","halt",
    "bow","unwilling"
2770 DATA "move","sword","willing",
    "status","axe","brave"
2780 DATA "rout","lance","valiant"
2790 DATA "knights","sergeants","men-at-
    arms","men-at-arms","archers","archers",
    "peasants","peasants"
2800 DATA "none","plains","jerkin",
    "village","chainmail","woods","plate",
    "hills"
2810 DATA 5,4,3,5,3,3,4,3,2,3,3,1,2,2,1,2,3,2,
    3,2,0,3,1,0
```

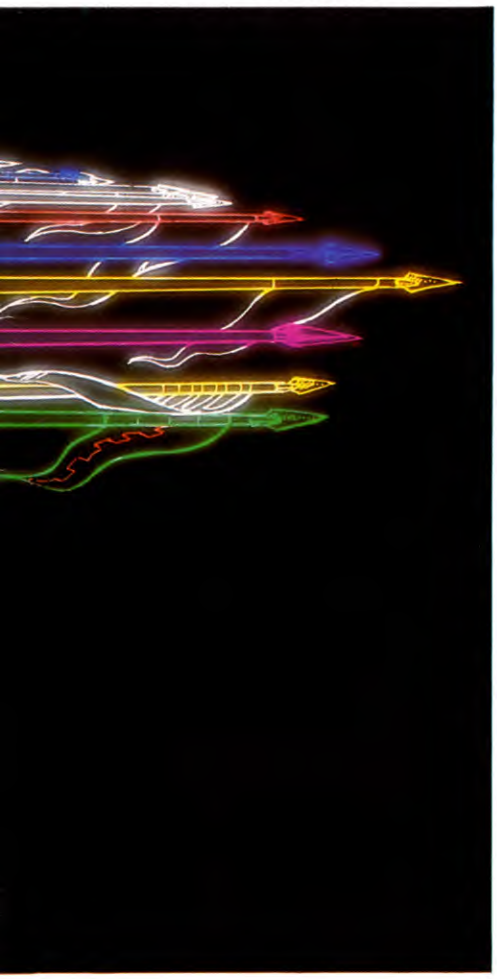
```
190 REM INITIALIZATION
195 DEF FNR(X)=INT(RND(1)*X)+1
200 VC=0:DE=0
210 POKE53281,5
220 PRINT CHR$(144)
340 PRINT "■"
360 DIM T$(7),O$(4),W$(4),M$(4),A$(3),
    R$(3)
380 FOR I=0 TO4:READ O$(I),W$(I),M$(I):
    NEXT I
390 FOR I=0 TO7:READ T$(I):NEXT I
400 FOR I=0 TO3:READ A$(I),R$(I):NEXT I
415 W$="NNYY"
420 RETURN
2870 DATA FIRE,NONE,COWARDLY,HALT,
    BOW,UNWILLING
2880 DATA MOVE,SWORD,WILLING,STATUS,
    AXE,BRAVE
2890 DATA ROUT,LANCE,VALIANT
2900 DATA KNIGHTS,SERGEANTS,MEN-AT-
    ARMS,MEN-AT-ARMS,ARCHERS,ARCHERS
2910 DATA PEASANTS,PEASANTS
2920 DATA NONE,PLAINS,JERKIN,VILLAGE,
    CHAINMAIL,WOODS,PLATE,HILLS
2930 DATA 4,3,3,4,2,3,3,2,2,2,2,1,1,1,1,1,2,2,
    2,1,0,2,0,0
```

```
190 DEF PROCinit
200 vic=FALSE:def=FALSE
360 DIM type$(7),order$(4),wpn$(4),mor$(4),
    arm$(3),ter$(3)
380 FOR i=0TO4:READ order$(i),wpn$(i),
    mor$(i):NEXT
390 FORi=0TO7:READ type$(i):NEXT
400 FORi=0TO3:READ arm$(i),ter$(i):NEXT
420 ENDPROC
2570 DATA fire,none,cowardly,halt,bow,
    unwilling,move,sword,willing,status,axe,
    brave,rout,lance,valiant,knights,sergeants,
    men-at-arms,men-at-arms,archers,archers,
```



```
peasants,peasants,none,plains,jerkin,village,
  chainmail,woods,plate,hills
2620 DATA 4,3,3,4,2,3,3,2,2,2,2,1,1,1,1,1,2,2,
  2,1,0,2,0,0
```

```
310 REM
330 COLOR 4,1
340 PCLS
360 DIM T$(8),O$(5),W$(5),M$(5),A$(4),
  R$(4)
380 FOR J=1 TO 5:READ O$(J),W$(J),
  M$(J):NEXT J
390 FOR J=1 TO 8:READ T$(J):NEXT J
400 FOR J=1 TO 4:READ A$(J),R$(J): NEXT
415 X$="NnYy"
420 RETURN
2760 DATA FIRE,NONE,COWARDLY,HALT,
  BOW,UNWILLING
2770 DATA MOVE,SWORD,WILLING,STATUS,
  AXE,BRAVE
2780 DATA ROUT,LANCE,VALIANT
2790 DATA KNIGHTS,SERGEANTS,MEN AT
  ARMS,MEN AT ARMS,ARCHERS,
  ARCHERS,PEASANTS,PEASANTS
2800 DATA NONE,PLAINS,JERKIN,VILLAGE,
  CHAINMAIL,WOODS,PLATE,HILLS
```



```
2810 DATA 5,4,3,5,3,3,4,3,2,3,3,1,2,2,1,2,3,2,
  3,2,0,3,1,0
```

Initial values are READ from the DATA lines, along with words which are equivalent to the various numeric values fed into the array. The words will be used in the text window to display the status of the battle clearly.

GIVING ORDERS

The whole game hinges on the player giving orders to the army—without this there would be no combat, and no winning or losing.

There are four orders you can give to your troops: Fire, Halt, Move and Status. The order to fire only applies to the archers, and they will fire on request even if there is no target around.

Giving orders to units has its own trade-offs. The more realistic the game, the more complex the giving of orders. The simpler the game, the easier it is to give orders. One result of this is that a realistic game will be very much more difficult to play; and if you wish to have a game that's easy to play, you'll have to make sacrifices in terms of nearness to reality. After you have written some wargames you'll probably formulate your own ideas on the best compromise.

The routine in Cavendish Field goes through each of the player's units in turn, identifying the unit to which orders can be given by changing its colour on the display. It also prints a message which asks if the standing orders for that unit (number x) are to be changed—the units continue doing whatever they were instructed by the last order issued to them until the player changes the order. If the player doesn't want to change the orders to that unit, all that needs to be done is to press the N key. If Y is pressed, then a menu of the four possible commands is given.

If Fire is selected, and the unit isn't archers, then another order is requested. Halt is self-explanatory. Move asks for a direction to be input. No check is made at this stage if a move is possible, so you may wish to modify the game so that no move option is displayed if no move is possible—but remember, adding input checks slows the game down. Status responds with a description of the current state of the unit.

SELECTING A UNIT

When it is the player's turn, the first unit is highlighted, followed by the other seven in turn. The unit which changes colour is the unit which the player must consider—should the orders be changed? As the unit is highlighted, prompts appear in the text window, along with the current orders.

```
1380 REM Select unit for orders
1390 GOSUB 2540
1400 INK 0
1410 PRINT FLASH1;AT T(i,8),T(i,9);u$(i)
1420 PRINT AT 17,0;"Unit number□";i;"□";
  t$(i);"□□□□□□"
1430 PRINT AT 18,0;"Current orders are
  to ";o$(T(i,1));"□";
1440 IF T(i,1)=3 THEN PRINT AT
  18,28;i$(T(i,2))
1450 REM Dummy for repeat loop
1460 PRINT AT 19,0;"Do you want to change
  orders (Y/N)?"
1465 LET y=0: LET y$=INKEY$
1466 IF y$="" THEN GOTO 1465
1470 FOR k=1 TO 4
1475 IF x$(k)=y$ THEN LET y=k
1480 NEXT k
1490 IF y=0 THEN GOTO 1450
1500 RETURN
```

```
1380 REM SELECT UNIT FOR ORDERS
1390 GOSUB 2540
1400 CL=6
1410 P=T(I,7):Q=T(I,8):GH=VAL(MID$
  (U$,I+1,1))+67:GOSUB2600
1420 PRINT "UNIT NUMBER□";I+1;"□";
  T$(I)
1430 PRINT "CURRENT ORDERS:□";O$
  (T(I,0));"□";
1440 IF T(I,0)=2 THEN PRINT
  MID$(I$,T(I,1),1)
1445 PRINT"□"
1450 PRINT"DO YOU WANT TO CHANGE
  ORDERS? (Y/N)"
1460 REM DUMMY FOR LOOP
1465 Y$="":GETY$
1470 FOR K=1 TO4
1475 IF MID$(W$,K,1)=Y$ THEN Y=K
1480 NEXT K
1490 IF Y=0 THEN 1460
1500 RETURN
```

```
1380 DEF PROCunitsel
1390 PROCclean
1400 COLOUR 0
1410 PRINT TAB(T%(i,7)+1,T%(i,8)+1);
  MID$(unst$,i+1,1)
1420 PRINT TAB(2,23);"Unit number□";i+1;
  "□";type$(i);"□□□□□□"
1430 PRINT TAB(2,24);"Current orders:□";
  order$(T%(i,0));"□";
1440 IF T%(i,0)=2 THEN PRINT
  MID$(dir$,T%(i,1),1)
1450 REPEAT
1460 PRINT TAB(2,25);"Change orders (Y/N)?"
1470 yn$=GET$
```

```
1480 yn=INSTR("NnYy",yn$)
1490 UNTIL yn
1500 ENDPROC
```

```
1380 REM SELECT UNIT FOR ORDERS
1390 GOSUB 2540
1400 COLOR 4
1410 DRAW"BM"+STR$(T(I,9)*8)+","+
     STR$(T(I,8)*8):UU=VAL(MID$(U$,I,1)):
     A$=UC$(UU):GOSUB 3030
1420 DRAW"BM0,144":A$="UNIT "+
     STR$(I)+"□"+T$(I)+"□□□□□
     □□":GOSUB 3190
1430 DRAW"BM0,152":A$="ORDERS ARE
     TO□□□"+O$(T(I,1))+"□":
     GOSUB 3190
1440 IF T(I,1)=3 THEN DRAW"BM160,152":
     A$=MID$(I$,T(I,2),1):GOSUB 3190
1450 REM
1460 DRAW"BM0,160":A$="CHANGE
     ORDERS□□Y OR N":GOSUB 3190
1465 Y=0:Y$=INKEY$
1466 IF Y$="" THEN 1465
1470 Y=INSTR(1,X$,Y$)
1490 IF Y=0 THEN 1450
1500 RETURN
3000 X9=PEEK(200):Y9=PEEK(202):LINE
     (X9,Y9)-(X9+7,Y9+7),PRESET,BF
3010 POKE 200,X9:POKE 202,Y9:DRAW A$
3020 RETURN
3030 X9=PEEK(200):Y9=PEEK(202):
     C9=PEEK(178):COLOR 2:LINE(X9,Y9)-
     (X9+7,Y9+7),PSET,BF
3040 POKE 178,C9:GOTO 3010
3050 REM DATA FOR LETTERS & DIGITS
3060 DATA BR4,BDD5RU3NR2U2ERFND5BR4
     BU,D6R3EUHEUHBR5,NR5D6R5BR3BU6,
     NR3D6R3EU4BUBR4,NR5D3NR3D3R5BR3
     BU6,NR5D3NR3D3BR8BU6,NR5D6R5U2
     BU4BR3
3070 DATA ND6D3R5D3U6BR3,R5L2D6L3R5
     BR3BU6,BD5RFREU5L2BR6,ND6D3R3FD2
     BU4U2BR4,D6R5BR3BU6,ND6DR5ND5
     UBR3,ND6DR2D3R3D2U6BR3
3080 DATA D6R5U6NL5BR3,D6U3R5U3NL5
     BR3,D6UR3FRBU2U4NL5BR3,D6U2R3FDU
     BU2NL2U3NL4BR4,NR5D3R5D3NL5BU6
     BR3,R5L2ND6BR5,D6R5U6BR3
3090 DATA D4RFDRUEU4BR4,D6UR5DU6
     BR3,D2RFGND2ERFND2HEU2BR4,D2RFD3
     RU3EU2BR4,R4D2GLGD2R4BU6BR3
3100 DATA BR3LGD4FREU4BR4BU,BR2DNL2
     D5L2R5BU6BR3,BDRERFDGL2D3R4BU6
     BR3,BDRERFDGFDGLHLBU5BR8,D4R5UD3
     BU6BR3,NR5D2R3FD2GLHLBU5BR8,BDB
     R5LHLGD2NR2D2FREUBU4BR4
3110 DATA R5D2LGLGD2BR7BU6,BR3LGDF
     GDFREUHEUBUBR4,BR3LGDFR2D2GLHL
     BU2BR4U2BUBR4
3120 REM SET UP CHARACTER ARRAYS
3130 DIM LE$(26)
3140 FOR K9=0 TO 26:READ LE$(K9):NEXT
3150 FOR K9=0 TO 9:READ NU$(K9):NEXT
3170 RETURN
3180 REM ROUTINE TO PRINT A$
3190 FOR K9=1 TO LEN(A$)
3200 B$=MID$(A$,K9,1)
3210 IF B$>="0" AND B$<="9" THEN
     DRAW NU$(VAL(B$)):GOTO 3240
3220 IF B$="□" THEN N9=0 ELSE
     N9=ASC(B$)-64
3230 DRAW LE$(N9)
3240 NEXT
3250 RETURN
```

The text window displays the current orders to that unit, and it changes colour on screen. The player is then asked DO YOU WANT TO CHANGE ORDERS (Y/N)?

The Dragon/Tandy program has additional lines (from 3000 to 3250) for drawing letters and numbers on the high resolution graphics screen.

ISSUING ORDERS

This routine displays the order options if the player wishes to change the orders given to a particular unit.

```
1900 REM Select Action
1910 GOSUB 2540
1920 PRINT AT 18,0;"Options are:"
1930 FOR j=1 TO 4
1940 PRINT AT 17+j,8; o$(j,1);
     " - ";o$(j)
1950 NEXT j
1960 REM Dummy for loop
1962 LET a=0
1965 LET f$="FfHhMmSs"
1970 LET g$=INKEY$: IF g$="" THEN
     GOTO 1970
1975 FOR k=1 TO 8
1980 IF f$(k)=g$ THEN LET a=INT
     ((k+1)/2)
1985 NEXT k
1990 IF a <=0 THEN GOTO 1960
2000 IF i<>6 AND i<>5 AND a=1
     THEN GOSUB 2540: PRINT AT 18,8;"No
     bows" : GOSUB 2410: GOTO 1910
2010 IF a=4 THEN GOSUB 2440: RETURN
2020 LET T(i,1)=a
2030 IF a=3 THEN GOSUB 2050
2040 RETURN
```

```
1900 REM SELECT ACTION
1910 GOSUB 2540
1920 PRINT "OPTIONS ARE:□"
1930 FOR J=0 TO 3
1940 PRINT LEFT$(O$(J),1);"□-□"; O$(J)
1950 NEXT J
1960 REM DUMMY FOR LOOP
1965 F$="FHMS"
1970 GET G$:IF G$="" THEN 1970
1973 A=-2
1975 FOR K=1 TO 4
1980 IF MID$(F$,K,1)=G$ THEN A=K-1
1985 NEXT K
1990 IF A<=-1 THEN 1960
2000 IF I<>4 AND I<>5 AND A=0
     THENGOSUB2540:PRINT "NO BOWS":
     GOSUB2410:GOTO 1910
2010 IF A=3 THEN GOSUB 2440:RETURN
2020 T(I,0)=A
2030 IF A=2 THEN GOSUB 2050
2040 RETURN
```

```
1900 DEF PROCactsel
1910 PROCclean
1920 PRINT TAB(2,23);"Options are:"
1930 FOR j=0TO3
1940 PRINT TAB(12,24+j); LEFT$(order$(j),
     1);"□-□";order$(j)
1950 NEXT
1960 REPEAT
1970 g$=GET$
1980 A%=(INSTR("FfHhMmSs",
     g$)+1) DIV 2-1
1990 UNTIL A%>-1
2000 IF i<>4 AND i<>5 AND A%=0
     THEN PRINT"No bows":GOTO 1920
2010 IFA%=3 THEN PROCstat: ENDPROC
2020 T%(i,0)=A%
2030 IF A%=2 THEN PROCww
2040 ENDPROC
```

```
1900 REM SELECT ACTION
1910 GOSUB 2540
1920 DRAW"BM0,152":A$="OPTIONS□□
     □□":GOSUB 3190
1930 FOR J=1 TO 4
1940 DRAW"BM104," + STR$(144+J*8):
     A$=LEFT$(O$(J),1)+"□□"+O$(J):
     GOSUB 3190
1950 NEXT J
1960 REM
1962 A=0
1965 F$="FHMS"
1970 G$=INKEY$:IF G$="" THEN 1970
1980 A=INSTR(1,F$,G$)
1990 IF A<=0 THEN 1960
2000 IF I<>6 AND I<> 5 AND A=1
     THEN GOSUB 2540:DRAW"BM64,152":
     A$="NO BOWS":GOSUB 3190:
     GOSUB 2410:GOTO 1910
2010 IF A=4 THEN GOSUB 2440: RETURN
2020 T(I,1)=A
2030 IF A=3 THEN GOSUB 2050
2040 RETURN
```

Lines 1920 to 1950 display the four options. Lines 1965 to 1985 read the keyboard, and set A (or a) equal to the order number—0 is Fire, 1 is Halt, 2 is Move and 3 is Status.

The Fire option will be dealt with in the next part of Cavendish Field. The Move and Status options will be dealt with below. There is no routine for Halt, because the Move routine is simply by-passed. The direction element in the troop array is not reset because, as you will see later, it is used in the hand-to-hand combat routine.

Line 2020 feeds the value of A into the troop array. The order number is placed in the first element of the array.

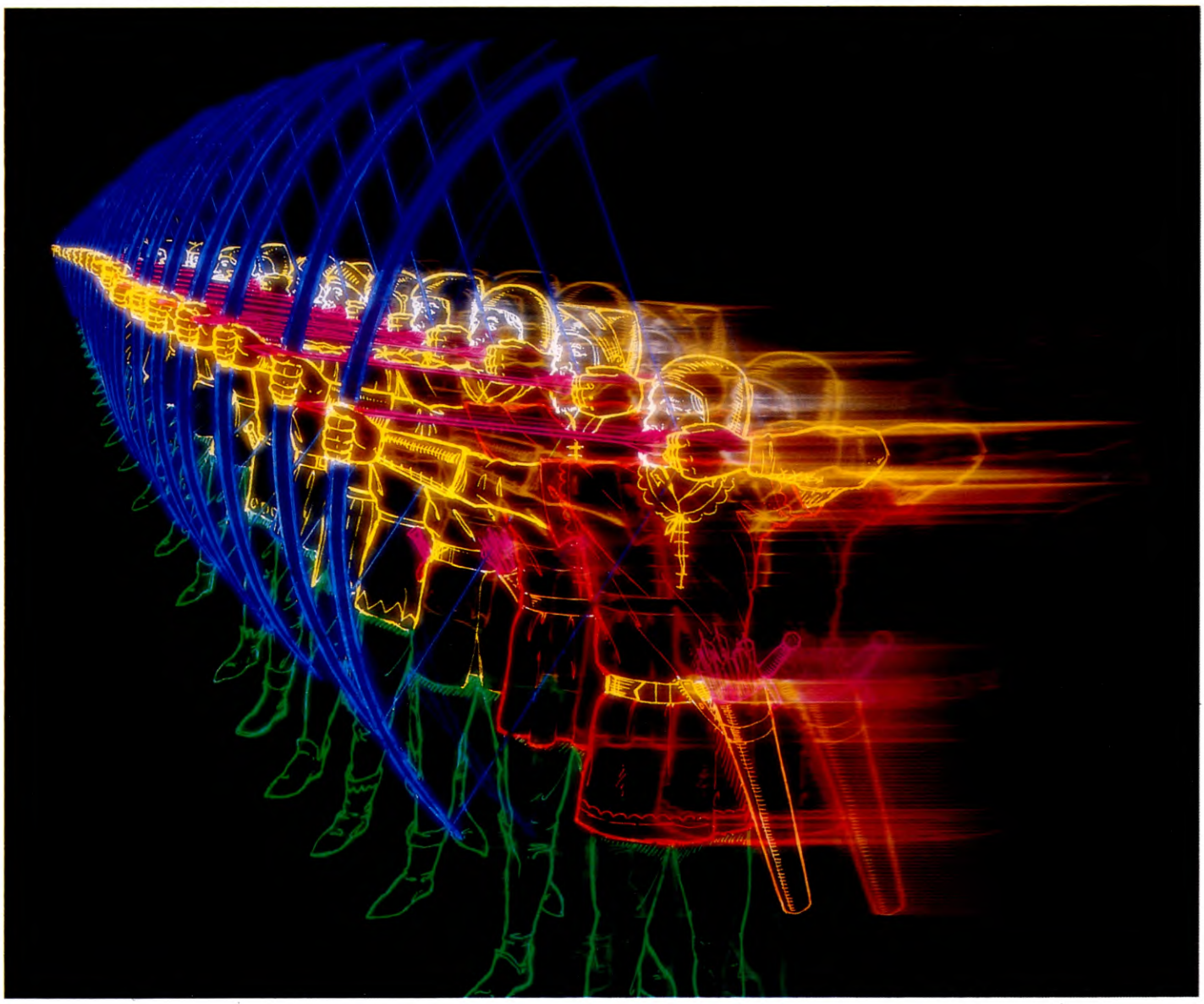
A NEW DIRECTION

If the player issues an order to move, a direction must be given also. This routine handles the movement options.

```
2050 REM Decide Movement Direction
2055 GOSUB 2540
2060 PRINT AT 17,0;"Which way (NSEW)?"
2065 LET g=0
2070 REM Dummy for loop
2080 LET g$=INKEY$: IF g$="" THEN
     GOTO 2080
2090 IF CODE (g$)>90 THEN LET
     g$=CHR$ (CODE (g$)-32)
2095 FOR k=1 TO 4
2100 IF i$(k)=g$ THEN LET g=k
2105 NEXT k
2110 IF g=0 THEN GOTO 2070
2120 LET T(i,2)=g
2130 RETURN
```

```
2050 REM DECIDE MOVEMENT DIRECTION
2055 GOSUB2540
2060 PRINT "WHICH WAY? (NSEW)"
2065 G=0
2070 REM DUMMY FOR REPEAT LOOP
2080 GET G$:IFG$=""THEN 2080
2090 IF ASC(G$)>90 THEN
     G$=CHR$(ASC(G$)-32)
2095 FOR K=1 TO 4
2100 IF MID$(I$,K,1)=G$ THEN G=K
2105 NEXT K
2110 IF G=0 THEN 2070
2120 T(I,1)=G
2130 RETURN
```

```
2050 DEF PROCww
2060 PRINT"Which way (NSEW)?"
2070 REPEAT
2080 g$=GET$
2090 IF ASC(g$)>90 THEN
```



```
    g$ = CHR$(ASC(g$) – 32)
2100 g = INSTR(dir$,g$)
2110 UNTIL g
2120 T%(i,1) = g
2130 ENDPROC
```

```
2050 REM DECIDE MOVEMENT DIRECTION
2055 GOSUB 2540
2060 DRAW"BM0,144":A$ = "WHICH WAY
     □□N□S□E□W":GOSUB 3190
2065 G = 0
2070 REM
2080 G$ = INKEY$:IF G$ = "" THEN 2080
2100 G = INSTR(1,I$,G$)
2110 IF G < = 0 THEN 2070
2120 T(I,2) = G
2130 RETURN
```

The routine checks that the letter input by the player is either N, S, E or W. If it is, then Line 2120 feeds the move direction code into the second element of the troop array.

STATUS

If you want to find out the status of any key unit while planning the big push, you can select the Status option.

```
2440 REM Unit status
2450 GOSUB 2540
2460 PRINT AT 17,0;"UNIT□";i;"□□□□
     Type:□";t$(i)
2470 PRINT AT 18,0;"Weapon:□";w$(T(i,3))
2480 PRINT AT 18,15;"Armour:□";a$(T(i,4))
2490 PRINT AT 19,0;"Strength:□";T(i,7)
2500 PRINT AT 19,14;"Attitude:";m$(T(i,5))
2510 PRINT AT 20,0;"Location:□";r$
     (m(T(i,8),T(i,9)) + 1)
2520 GOSUB 2410
2530 RETURN
```

```
2440 REM STATUS OF UNIT
2450 GOSUB 2540
2460 PRINT"UNIT□";I + 1;
     "□TYPE:□";T$(I)
2470 PRINT "WEAPON:□";W$(T(I,2));
     "□□□□□□□ARMOUR:□";A$
     (T(I,3))
2480 PRINT "STRENGTH:□";T(I,6)
2500 PRINT "LOCATION:□";R$(M(T(I,7),
     T(I,8)));"□□□ATTITUDE:□";
     M$(T(I,4))
2520 GOSUB 2410
2530 RETURN
```

```
2440 DEF PROCstat
2450 PROCclean
2460 PRINTTAB(2,23); "Unit□"; i + 1;
     "□□□□□ Type: □"; type$(i)
2470 PRINTTAB(2,24); "Weapon: □"; wpn$
     (T%(i,2))
2480 PRINTTAB(2,25); "Armour: □"; arm$
     (T%(i,3))
2490 PRINTTAB(2,26); "Current strength: □";
     T%(i,6)
2500 PRINTTAB(2,27); "Located in □"; ter$
     (FNmread (T% (i,7), T%(i,8)))
2510 PRINTTAB(2,28); "Attitude is □"; mor$
     (T%(i,4))
2520 PROCpause
2530 ENDPROC
```

```
2440 REM UNIT STATUS
2450 GOSUB 2540
2460 DRAW"BM0,144":A$ = "UNIT" + STR$
     (I) + "□□□□□TYPE□" + T$(I):
     GOSUB 3190
2470 DRAW"BM0,152":A$ = "WEAPON
     □" + W$(T(I,3)):GOSUB 3190
2480 DRAW"BM120,152":A$ = "ARMOUR
     □" + A$(T(I,4)):GOSUB 3190
2490 DRAW"BM0,160":A$ =
     "STRENGTH" + STR$(T(I,7)):GOSUB 3190
2500 DRAW"BM120,160":A$ = "ATTITUDE
     □" + M$(T(I,5)):GOSUB 3190
2510 DRAW"BM0,168":A$ = "LOCATION
     □" + R$(M(T(I,8),T(I,9)) + 1):
     GOSUB 3190
2520 GOSUB 2410
2530 RETURN
```

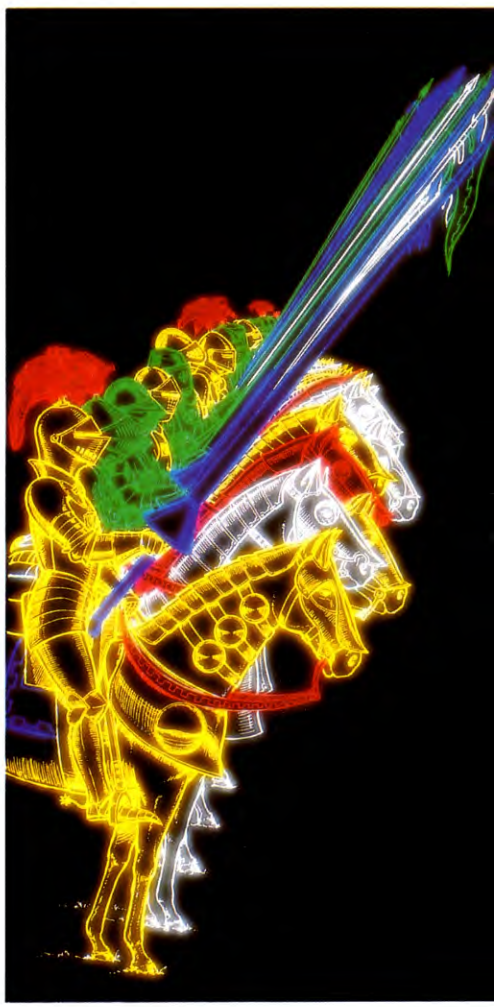
All the elements in the troop array (or their equivalents in words) are displayed.

THE EFFECT OF ORDERS

The routine tells the player if a particular unit is complying with the order.

```
1020 REM Effect of orders
1030 FOR i = 1 TO 16
1032 IF t(i,1) > 3 THEN GOTO 1140
1035 INK 0: GOSUB 2540: PRINT AT
     17,0;"Unit□";i;"□decides to act"
1040 LET cl = 1: IF i > 8 THEN LET cl = 2: INK
     cl
1050 IF T(i,1) = 3 THEN LET b = i: GOSUB
     1160
1055 IF T(i,1) = 2 THEN GOTO 1140
1060 IF T(i,1) = 1 THEN LET sh = i: GOSUB
     1710: GOTO 1140
1070 FOR f = –1 TO 1
1080 FOR g = –1 TO 1
1090 FOR e = 1 TO 16
1100 IF (T(i,8) + f = T(e,8)) AND
     (T(i,9) + g = T(e,9)) AND T(e,1) < > 5
     THEN LET us = i: LET th = e: GOSUB 1510
1110 NEXT e
1120 NEXT g
1130 NEXT f
1140 NEXT i
1150 RETURN
```

```
1020 REM EFFECT OF ORDERS
1030 FOR I = 0 TO 15
1032 IF T(I,0) > 2 THEN 1140
1034 GOSUB2540:PRINT"UNIT□";I + 1;"□
     DECIDES TO ACT"
1040 CO = 1:IF I > 7 THEN CO = 9
1050 IF T(I,0) = 2 THEN B = I:GOSUB 1160
1055 IF T(I,0) = 1 THEN 1140
1060 IF T(I,0) = 0 THEN SH = I:GOSUB 1710:
     GOTO 1140
1070 FOR F = –1 TO 1
1080 FOR G = –1 TO 1
1090 FOR E = 0 TO 15
1100 IF(T(I,7) + F = T(E,7))AND(T(I,8) +
     G = T(E,8))ANDT(E,0) < > 4THEN US = I:
     TH = E:GOSUB1510
1110 NEXT E
1120 NEXT G
1130 NEXT F
1140 NEXT I
1150 RETURN
```



```
1020 DEF PROCeffect
1030 FOR i=0 TO 15
1031 COLOUR 0
1032 IF T%(i,0)>2 THEN 1140
1035 PROCclean:PRINT TAB(2,23);"Unit □";
     i+1;"□decides to act"
1040 cl=(i□DIV 8)+1:COLOURcl
1050 IF T%(i,0)=2 THEN PROCmove(i)
1055 IF T%(i,0)=1 THEN GOTO1140
1060 IF T%(i,0)=0 THEN PROCfire(i):
     GOTO 1140
1070 FORfig=-1TO1
1080 FORgif=-1TO1
1090 FOREn=0TO15
1100 IFT%(i,7)+fig=T%(en,7)AND T%
     (i,8)+gif=T%(en,8) AND T%(en,
     0)<>4 THENPROCcombat(i,en)
1110 NEXT:NEXT:NEXT
1140 NEXT
1150 ENDPROC
```

```
1020 REM EFFECT OF ORDERS
1030 FOR I=1 TO 16
1032 IF T(I,1)>3 THEN 1140
1035 COLOR 4:GOSUB 2540:DRAW
     "BM0,144":A$="UNIT"+STR$(I)+
     "□ACTS":GOSUB 3190
1040 CL=3: IF I>8 THEN CL=4:COLOR CL
1050 IF T(I,1)=3 THEN B=I:GOSUB 1160
1055 IF T(I,1)=2 THEN 1140
1060 IF T(I,1)=1 THEN SH=I:
     GOSUB 1710:GOTO 1140
1070 FOR F=-1 TO 1
1080 FOR G=-1 TO 1
1090 FOR E=1 TO 16
1100 IF (T(I,8)+F=T(E,8)) AND (T(I,9)+
     G=T(E,9)) AND T(E,1)<>5 THEN
     US=I:TH=E:GOSUB 1510
1110 NEXT E,G,F
1140 NEXT I
1150 RETURN
```

The routine operates quite simply. It cycles through each of the sixteen units and acts on its order. It will only allow combat if a unit has move orders, because that unit is then the attacker. If it finds halt orders, it moves on to the next unit, taking no action. If it finds fire orders, it calls the missile routine. If it finds move orders it calls the movement routine.

When movement has been carried out it looks at each map square adjacent to the unit's new position—Lines 1070 to 1150. If the adjacent square is occupied by an enemy unit the combat routine is called—you will see how to enter this in the next part.

THE COMPUTER OPPONENT

The computer gives orders to its units in a more or less random way. In the last part of this series of articles you'll see how to give your opponent some degree of intelligence, but for the moment, this simplified routine will give you a playable wargame.

```
2140 REM Enemy selects action
2150 LET T(e,2)=3
2160 LET T(e,1)=FN r(3)
2170 IF T(e,1)=1 AND T(e,3)<>2 THEN
     GOTO 2160
2180 IF T(e,1)=3 THEN IF FN r(2)=1 THEN
     LET T(e,2)=FN r(4)
2190 RETURN
```

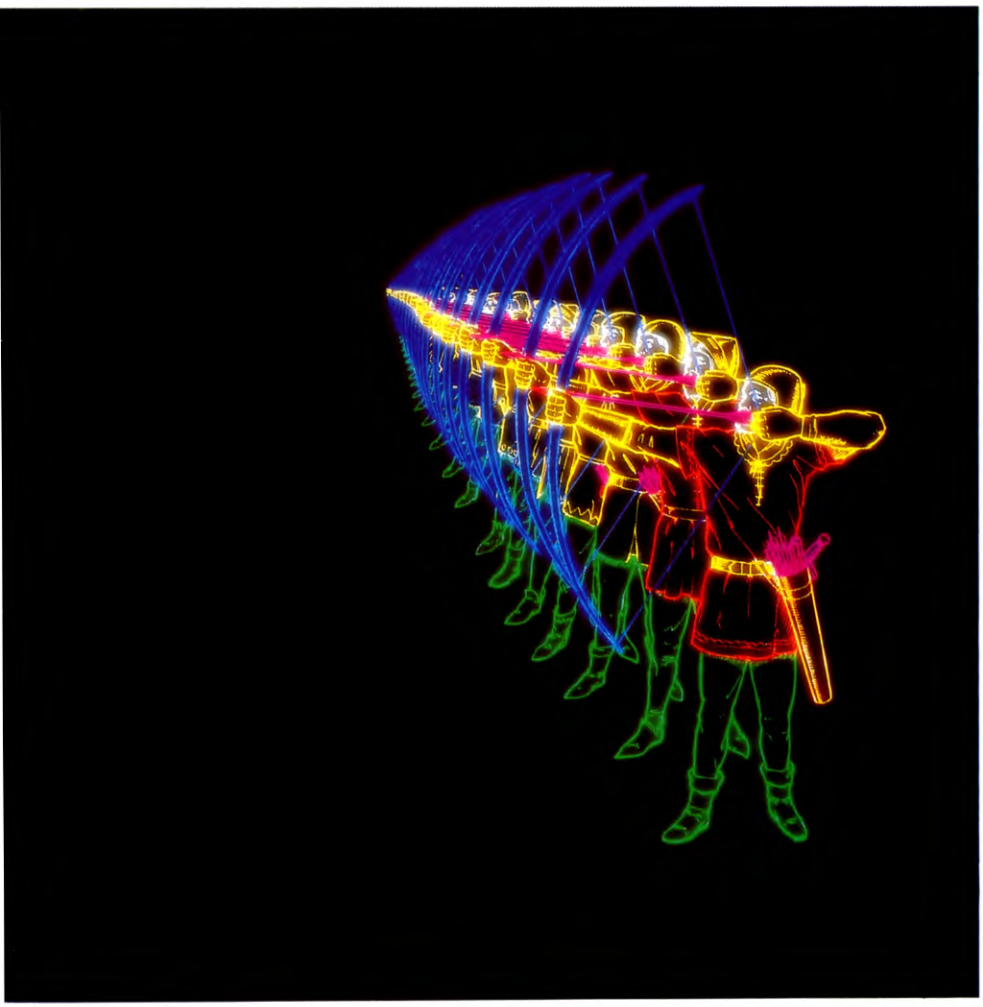
```
2140 REM SELECT ENEMY ACTION
2150 T(E,1)=3
2160 T(E,0)=FNR(3)-1
2170 IF T(E,0)=0 AND T(E,2)<>1 THEN
     2160
2180 IF T(E,0)=2 THEN IF FNR(2)=1 THEN
     T(E,1)=FNR(4-1)
2190 RETURN
```

```
2140 DEF PROCensl
2150 T%(en,1)=3
2160 T%(en,0)=RND(3)-1
2170 IF T%(en,0)=0 AND T%(en,2)<>1
     THEN 2160
2180 IF T%(en,0)=2 THEN IF RND(2)=1
     THEN T%(en,1)=RND(4)-1
2190 ENDPROC
```

```
2140 REM ENEMY SELECTS ACTION
2150 T(E,2)=3
2160 T(E,1)=RND(3)
2170 IF T(E,1)=1 AND T(E,3)<>2 THEN
     2160
2180 IF T(E,1)=3 AND RND(2)=1 THEN
     T(E,2)=RND(4)
2190 RETURN
```

The routine generates a random number in Line 2160 which decides on the action to be taken by each of the computer's units in turn.

If movement is selected, the movement isn't completely random because the units will tend to move south—see Line 2150.



WARGAMING: INTO BATTLE!

Here comes the crunch. At last combat can begin, with routines for missile combat and hand-to-hand fighting. Test morale on the beaten side and tot up the casualties

Cavendish Field is now almost completed; you can give your troops orders and move them about the battlefield. What is missing is the combat routines, and when you have entered these you will be able to try out your game.

What happens when the two sides engage in combat can be very complex, but again, decisions have to be made about what is going to go into your game. In the simplest case of combat, you could say something like: 'If two units meet, then the biggest one wins.' This assumes that the relative sizes of the units is the deciding factor in combat. Or you may decide that the outcome depends on the level of morale, or the number of surviving horses, or whatever. No-one really knows what wins battles, so in any wargame it is a matter of choice and program design to decide which factors are important.

MISSILES

In Cavendish Field there are two different kinds of combat—missile (arrows) and hand-to-hand. This first routine deals with missile combat.

```
1710 REM Missile Routine
1720 GOSUB 2540
1730 PRINT AT 18,0;"Unit□";
     sh;"□fires"
1740 LET fx=5: LET fy=5: LET gp=-1
1745 LET st=9
1750 IF sh>8 THEN LET st=1
1770 FOR m=st TO (st+7)
1780 LET tm=ABS (T(m,8)-T(sh,8)): LET
     ty=ABS (T(m,9)-T(sh,9))
1785 IF tm<fx AND T(m,1)<5 AND ty<fy
     THEN LET fx=tm: LET fy=ty: LET gp=m
1790 NEXT m
1800 IF gp=-1 THEN PRINT AT 19,0;
     "Nothing in range" : GOSUB 2410:
     RETURN
1810 LET C=8-T(gp,4)- ABS (fx-fy)
1820 IF gp<3 OR gp=9 OR gp=10 THEN
     LET C=C+1
1830 IF m(T(gp,8),T(gp,9))=2 THEN LET
     C=C-2
1840 IF T(gp,1)<>2 THEN LET C=C+1
1850 LET C=(C+(INT (T(sh,7)/40))+ FN
     r(3))*10
1860 LET T(gp,7)=T(gp,7)-C
1870 PRINT "That causes□";C;
     "□casualties on unit□";gp
1875 GOSUB 2410
1880 LET un=gp: GOSUB 2200
1890 RETURN
```

```
1710 REM MISSILE ROUTINE
1720 GOSUB 2540
1730 PRINT"UNIT□";SH+1;"□FIRES"
1740 FX=5:FY=5:GP=-1
1745 BG=8
1750 IF SH>7 THEN BG=0
1770 FOR M= BG TO (BG+7)
1780 TM=ABS(T(M,7)-T(SH,7)):TY=ABS
     (T(M,8)-T(SH,8))
1785 IF TM<FX AND TY<FY AND
     T(M,0)<4 THEN FX=TM:FY=TY:GP=M
1790 NEXT M
1800 IF GP=-1 THEN PRINT "NOTHING
     IN RANGE":GOSUB2410:RETURN
1810 C=8-T(GP,3)-ABS(FX-FY)
1820 IF GP<2 OR GP=8 OR GP=9 THEN
     C=C+1
1830 IF M(T(GP,7),T(GP,8))=2 THEN
     C=C-2
1840 IF T(GP,0)<>2 THEN C=C+1
1850 C=(C+(INT(T(SH,6)/40))+FNR(3))*
     10
1860 T(GP,6)=T(GP,6)-C
1870 PRINT "UNIT□";GP+1;"□SUFFERS
     □";C;"□CASUALTIES"
1875 GOSUB 2410
1880 UN=GP:GOSUB 2200
1890 RETURN
```

```
1710 DEF PROCfire(sht)
1720 PROCclean
1730 PRINT TAB(2,23);"Unit□";sht+1;"□
     fires"
1740 fx=5:fy=5:G%=-1
1750 IF sht>7 THEN st=0 ELSE st=8
1770 FOR m=st TO (st+7)
1780 IF ABS(T%(m,7)-T%(sht,7))<fx□
     AND ABS(T%(m,8)-T%(sht,8))<fy□
     AND T%(m,0)<4 THEN fx=ABS
     (T%(m,7)-T%(sht,7)):fy=ABS(T%
     (m,8)-T%(sht,8)):G%=m
1790 NEXT
1800 IF G%=-1 THEN PRINT TAB(2,24);
     "Nothing in range":PROCpause:ENDPROC
1810 C%=8-T%(G%,3)-ABS(fx-fy)
1820 IF G%<2 OR G%=8 OR G%=9 THEN
     C%=C%+1
1830 IF FNmread(T%(G%,7),T%(G%,8))=2
     THEN C%=C%-2
1840 IF T%(G%,0)<>2 THEN C%=C%+1
1850 C%=(C%+(T%(sht,6) DIV
     40)+RND(3))*10
1860 T%(G%,6)=T%(G%,6)-C%
1870 PRINT"That causes□";C%;
     "□casualties on unit□";
     G%+1:PROCpause
1880 PROCmorale(G%)
1890 ENDPROC
```

```
1710 REM MISSILE ROUTINE
1720 GOSUB 2540
1730 DRAW"BM0,152":A$="UNIT"+
     STR$(SH)+"□FIRES":GOSUB 3190
1740 FX=5:FY=5:GP=-1
1745 ST=9
1750 IF SH>8 THEN ST=1
1770 FOR M=ST TO (ST+7)
1780 TM=ABS(T(M,8)-T(SH,8)):TY=ABS
     (T(M,9)-T(SH,9))
1785 IF TM<FX AND T(M,1)<5 AND
     TY<FY THEN FX=TM:
     FY=TY:GP=M
1790 NEXT M
1800 IF GP=-1 THEN DRAW"BM0,160":
     A$="NOTHING IN RANGE":GOSUB
     3190:GOSUB 2410:RETURN
1810 C=8-T(GP,4)-ABS(FX-FY)
1820 IF GP<3 OR GP=9 OR GP=10 THEN
     C=C+1
1830 IF M(T(GP,8),T(GP,9))=2 THEN
     C=C-2
1840 IF T(GP,1)<>2 THEN C=C+1
1850 C=(C+(INT(T(SH,7)/40))+
     RND(3))*10
1860 T(GP,7)=T(GP,7)-C
1870 DRAW"BM0,160":A$="CAUSES"+
     STR$(C)+"□CASUALTIES ON UNIT"+
     STR$(GP):GOSUB 3190
1875 GOSUB 2410
1880 UN=GP:GOSUB 2200
1890 RETURN
```

The missile combat routine is called every time the archers are ordered to open fire—in other wargames you might have more than one kind of unit which can open fire.

When a unit is ordered to fire G% (or gp or GP) is set to −1 in Line 174Ø. Next, the routine will decide if there is a suitable target. The coordinates of the shooting unit are compared with the coordinates of each of the enemy units. Line 178Ø compares the difference between the shooter's position and the target's. If the target is within a five-square range, G% (or gp or GP) is set to the target unit's number. If more than one target is within range, the nearest unit will be remembered.

If no target is found, G% (or gp or GP) will remain at −1, and the player will receive a NOTHING IN RANGE message. The routine then ends.

If the routine finds a target, the casualties are worked out. Casualties are stored in C% (or C).

Several factors affect whether losses are heavy or light. First, Line 181Ø starts with a value of 8 and subtracts the armour class of the target, and then a factor for range. Line 182Ø checks if the target is cavalry, and if it is, adds 1 to C% (or C). Line 183Ø checks if the target is in cover (terrain = type 2), then 2 is subtracted. Line 184Ø checks if the target is moving (recorded as the current order not being Halt) in which case 1 is added (archery attacks are more effective if the unit is charging). Finally, Line 185Ø adds the result to one fortieth of the strength of the shooting

unit, adds a small random number, and multiplies the result by ten. The result is the number of casualties suffered by the target unit. The casualties are subtracted from the current strength of the target unit, and a routine to test morale is called.

THE CLASH

Hand-to-hand combat is calculated in a similar way.

```
1510 REM Combat
1520 IF (us<9 AND th<9) OR (us>8 AND
   th>8) THEN RETURN
1530 IF T(us,1)=5 OR T(th,1)=5 THEN
   RETURN
1540 GOSUB 2540
1550 PRINT AT 18,0;"Combat!!!"
1560 LET at=INT ((T(us,7)-
   T(th,7))/50)
1570 LET at=at+T(us,3)-T(th,4)+T(us,
   5)+FN r(5)
1580 IF ABS (T(us,2)-T(th,2))<>2 THEN
   LET at=at+2
1590 IF us<3 OR us=9 OR us=10 THEN
   LET at=at+1
1600 LET dr=INT ((T(th,7)-T(us,7))/60)
1610 LET dr=dr+T(th,3)-T(us,4)-T(th,
   5)+m(T(th,8),T(th,9))+FN r(3)+2
1615 LET wn=th: LET lo=us
1620 IF at>dr THEN LET wn=us: LET lo=th
1630 LET wc=INT (T(wn,7)/10): IF wc<1
   THEN LET wc=1
1640 LET T(wn,7)=T(wn,7)-wc
1650 LET lc=INT (T(lo,7)/5): IF lc<1 THEN
   LET lc=1
1660 LET T(lo,7)=T(lo,7)-lc
1670 PRINT wn;" loses ";wc;" ";lo;" loses ";lc
1680 GOSUB 2410
1690 LET un=lo: GOSUB 2200
1700 RETURN
```

```
1510 REM COMBAT
1520 IF(US<8 AND TH<8)OR(US>7 AND
   TH>7)THEN RETURN
1530 IF T(US,0)=4 OR T(TH,0)=4 THEN
   RETURN
1540 GOSUB 2540
1550 PRINT "COMBAT!!!"
1560 AT=INT((T(US,6)-T(TH,6))/50)
1570 AT=AT+T(US,2)-T(TH,3)+
   T(US,4)+FNR(5)
1580 IF ABS(T(US,1)-T(TH,1))<>2 THEN
   AT=AT+2
1590 IF US<2 OR US=8 OR US=9 THEN
   AT=AT+1
1600 DF=INT((T(TH,6)-T(US,6))/60)
1610 DF=DF+T(TH,2)-T(US,3)+T(TH,
   4)+M(T(TH,7),T(TH,8))+FNR(3)+2
1615 WN=TH:LO=US
1620 IF AT>DF THEN WN=US:LO=TH
1630 WC=INT(T(WN,6)/10):
   IF WC<1 THEN WC=1
1640 T(WN,6)=T(WN,6)-WC
1650 LC=INT(T(LO,6)/5):IF LC<1 THEN
   LC=1
1660 T(LO,6)=T(LO,6)-LC
1670 PRINT WN+1;" LOSES ";WC;
   " ";LO+1;" LOSES ";LC
1680 GOSUB 2410
1690 UN=LO: GOSUB 2200
1700 RETURN
```

```
1510 DEF PROCcombat(us,th)
1520 IF (us<8 AND th<8) OR (us>7 AND
     th>7) THEN ENDPROC
1530 IF T%(us,0)=4 OR T%(th,0)=4 THEN
     ENDPROC
1540 PROCclean
1550 PRINT TAB(2,23);"Combat!!!"
1560 att=(T%(us,6)-T%(th,6)) DIV 50
1570 att=att+T%(us,2)-T%(th,3)+T%(us,
     4)+RND(5)
1580 IF ABS(T%(us,1)-T%(th,1))<>2
     THEN att=att+2
1590 IF us<2 OR us=8 OR us=9 THEN
     att=att+1
1600 def=(T%(th,6)-T%(us,6)) DIV 60
1610 def=def+T%(th,2)-T%(us,3)+T%
     (th,4)+FNmread(T%(th,7),T%(th,8))+
     RND(3)+2
1620 IF att>def THEN win=us:
     lose=th□ELSE win=th:lose=us
1630 wc=T%(win,6) DIV 10:IF wc<1 THEN
     wc=1
1640 T%(win,6)=T%(win,6)-wc
1650 lc=T%(lose,6) DIV 5:IF lc<1 THEN
     lc=1
1660 T%(lose,6)=T%(lose,6)-lc
1670 PRINT win+1;"□loses□";wc;"□"
     lose+1;"□loses□";lc
1680 PROCpause:PROCmorale(lose)
1700 ENDPROC
```

```
1510 REM COMBAT
1520 IF (US<9 AND TH<9) OR (US>8
     AND TH>8) THEN RETURN
1530 IF T(US,1)=5 OR T(TH,1)=5 THEN
     RETURN
1540 GOSUB 2540
1550 DRAW"BM0,144":A$="COMBAT
     □□□":GOSUB 3190
1560 AT=INT((T(US,7)-T(TH,7))/50)
1570 AT=AT+T(US,3)-T(TH,4)+T
     (US,5)+RND(5)
1580 IF ABS(T(US,2)-T(TH,2))<>2 THEN
     AT=AT+2
1590 IF US<3 OR US=9 OR US=10 THEN
     AT=AT+1
1600 DF=INT((T(TH,7)-T(US,7))/60)
1610 DF=DF+T(TH,3)-T(US,4)+
     T(TH,5)+M(T(TH,8),T(TH,9))+RND
     (3)+2
1615 WN=TH:LO=US
1620 IF AT>DF THEN WN=US:LO=TH
1630 WC=INT(T(WN,7)/10):IF WC<1 THEN
     WC=1
1640 T(WN,7)=T(WN,7)-WC
1650 LC=INT(T(LO,7)/5):IF LC<1 THEN
     LC=1
1660 T(LO,7)=T(LO,7)-LC
1670 DRAW"BM0,160":A$=STR$(WN)+
     "□LOSES"+STR$(WC)+"□"+STR$
     (LO)+"□LOSES"+STR$(LC):
     GOSUB 3190
1680 GOSUB 2410
1690 UN=LO:GOSUB 2200
1700 RETURN
```

The routine is called every time two units meet and RETURNs, or ENDPROCs on the Acorns, immediately if the two units belong to the same army. If the unit is being routed, Line 1530 RETURNs also.

The main difference between hand-to-hand combat and missile combat is that both sides have something to say in the affray. This means that hand-to-hand combat is a messier business requiring two sets of casualties to be calculated.

In Line 1560 the attackers start with one fiftieth of the difference between the two sides. Next, the difference between the attacker's weapon and the defender's armour is added, along with the value of the attacker's morale, and a random number up to five.

The attackers also gain a bonus if the enemy is not directly facing them. This will be the case either if the unit is not moving directly towards the attacker, or (when the unit has halted) if it was not moving in that direction last time it moved. This is why the 'direction' element of the troop array is never reset, and must always hold a direction. Historically, it seems that attacking the rear or flank was one of the most significant of combat factors. Finally, the attackers get a bonus of one point if they are cavalry.

Defenders are treated similarly. Starting at Line 1600, they lose one sixtieth of the difference in numbers, plus the difference between the defender's weapon class and the attacker's armour class, plus the defender's morale, plus a bonus for the terrain they are defending, plus a fixed bonus of 2 and a random factor of up to 3. The random factor is intended to represent the fact that defending is easier than attacking, but sometimes the bloodlust of the attackers can intervene to overpower the innate advantage of defending.

The attackers will now have a value (at), as will the defenders (df). The side with the largest value wins that round of that fight, and therefore, loses only one tenth of its strength in casualties. The lower's forces are reduced by one fifth. Finally, the morale test is called for the loser.

THE MORALE MINORITY

The psychological factor is very important in warfare. It is very unlikely that an army will win a battle, equipped with the most powerful tanks in the world, if the troops hate their generals, think that the enemy have a just cause, or don't like the idea of fighting anyway. There are thousands of factors involved in the individual psychology of soldiers, and no game has ever come close to representing them all. At the most ridiculous level, the attitude of a warrior to the current fight can

depend on whether he had a good night's sleep, or not, or on the quality of his last meal.

Not only can morale be complicated, but it can effect just about every aspect of the battle, giving a boost, or hampering the sides. In Cavendish Field, morale is only tested when a unit loses a fight, or is fired upon.

```
2200 REM Unit Morale Test
2210 IF T(un,6) – T(un,7) < ((T(un,6)/100)*
   ((T(un,5) + 2))) THEN RETURN
2220 GOSUB 2540
2230 PRINT AT 18,0;"Losses are too great."
2240 PRINT AT 19,0;"Unit□";
   un;"□disintegrates"
2250 GOSUB 2410
2260 LET T(un,1) = 5
2270 PRINT AT T(un,8),T(un,9);"□"
2280 RETURN
```

```
2200 REM UNIT MORALE TEST
2210 IF T(UN,5) – T(UN,6) < ((T(UN,5)/100)*
   ((T(UN,4) + 2)*10)) THEN RETURN
2220 GOSUB 2540
2230 PRINT "LOSSES ARE TOO GREAT"
2240 PRINT "UNIT□";UN + 1;"□
   DISINTEGRATES"
2250 GOSUB 2410
2260 T(UN,0) = 4
2270 P = T(UN,7):Q = T(UN,8):
   GH = 32:GOSUB 2600
2280 RETURN
```

```
2200 DEF PROCmorale(un)
2210 IF T%(un,5) – T%(un,6) < ((T%(un,5)/
   100)*((T%(un,4) + 2)*10)) THEN
   ENDPROC
2220 PROCclean
2230 PRINT TAB(2,23);"Losses are too great."
2240 PRINT TAB(2,24);"Unit□";un + 1;"□
   disintegrates"
2250 PROCpause
2260 T%(un,0) = 4
2270 PRINT TAB(T%(un,7) + 1,T%(un,8) + 1);
   "□"
2280 ENDPROC
```

```
2200 REM UNIT MORALE TEST
2210 IF T(UN,6) – T(UN,7) < ((T(UN,6)/100)*
   ((T(UN,5) + 2)*10)) THEN RETURN
2220 GOSUB 2540
2230 DRAW"BM0,152":A$ = "LOSSES ARE
   TOO GREAT":GOSUB 3190
2240 DRAW"BM0,160":A$ = "UNIT" + STR$
   (UN) + "□DISINTEGRATES":GOSUB 3190
2250 GOSUB 2410
2260 T(UN,1) = 5
2270 X9 = T(UN,9)*8:Y9 = T(UN,8)*8:LINE
   (X9,Y9) – (X9 + 7,Y9 + 7),PRESET,BF
2280 RETURN
```

Line 2210 subtracts the current strength from the initial strength and compares it with the base morale of the unit. If the unit has suffered 30% casualties and is cowardly, it disintegrates and takes no further part in the battle. On the other hand, a unit which has a higher morale can suffer up to 70% casualties before disintegrating.

If you feel the routine is too clear-cut, you may want to extend it by adding a small random factor, considering if the unit was in cover, or examining the overall situation (how many enemy units are left, how many friendly units are nearby?).

When a unit fails the morale test, a message to that effect is displayed by Lines 2230 and 2240. The command element of the array is set to five (or four in the case of the Acorn and Commodore), which means that the unit has been routed. The unit is blanked from the screen, and ignored in all further requests for orders and combat. However, because there are deserters milling all over the battlefield, the unit can still be an obstacle to movement, all the more hazardous because it is 'invisible' to the commander.

THE ARMISTICE

There are a few finishing touches to be added to the program: first, a victory condition.

```
2290 REM Test for victory
2300 LET gd = 0: LET bd = 0
2310 FOR m = 1 TO 8
2320 IF T(m,1) < > 5 THEN LET gd = gd + 1
2330 IF T(m + 8,1) < > 5 THEN LET
   bd = bd + 1
2340 NEXT m
2350 IF gd > bd*2 OR (bd < 2 AND gd > 2)
   THEN LET vc = 1
2360 IF bd > gd*2 OR (gd < 2 AND bd > 2)
   THEN LET de = 1
2370 RETURN
2380 REM End Message
2390 IF vc = 1 THEN PRINT "VICTORY!!"
2395 IF de = 1 THEN PRINT "A crushing
   defeat"
2400 RETURN
```

```
2290 REM TEST FOR VICTORY
2300 GD = 0:BD = 0
2310 FOR M = 0 TO7
2320 IF T(M,0) < > 4 THEN GD = GD + 1
2330 IF T(M + 8,0) < > 4 THEN BD = BD + 1
2340 NEXT M
2350 IF GD > BD*2 OR (BD < 2 AND GD > 2)
   THEN VC = 1
2360 IF BD > GD*2 OR (GD < 2 AND BD > 2)
   THEN DE = 1
2370 RETURN
2380 REM MESSAGE
2390 IF VC = 1 THEN PRINT "VICTORY!!"
2395 IF DE = 1 THEN PRINT "A CRUSHING
   DEFEAT"
2400 RETURN
```

```
2290 DEF PROCvicdef
2300 gd = 0:bd = 0
2310 FOR m = 0TO7
2320 IF T%(m,0) < > 4 THENgd = gd + 1
2330 IF T%(m + 8,0) < > 4 THEN bd = bd + 1
2340 NEXT
2350 IF gd > bd*2 OR (bd < 2 AND gd > 2)
   THEN vic = TRUE
2360 IF bd > gd*2 OR (gd < 2 AND bd > 2)
   THEN def = TRUE
2370 ENDPROC
2380 DEF PROCmess
2390 IF vic = TRUE THEN PRINT "VICTORY!"
   ELSE PRINT"A crushing defeat"
2400 ENDPROC
```

```
2290 REM TEST FOR VICTORY
2300 GD = 0:BD = 0
2310 FOR M = 1 TO 8
2320 IF T(M,1) < > 5 THEN GD = GD + 1
2330 IF T(M + 8,1) < > 5 THEN BD = BD + 1
2340 NEXT M
2350 IF GD > BD*2 OR (BD < 2 AND GD > 2)
   THEN VC = 1
2360 IF BD > GD*2 OR (GD < 2 AND BD > 2)
   THEN DE = 1
2370 RETURN
2380 REM END MESSAGE
2390 IF VC = 1 THEN DRAW"BM0,176":A$ =
   "VICTORY":GOSUB 3190
2395 IF DE = 1 THEN DRAW"BM0,176":A$ =
   "A CRUSHING DEFEAT":GOSUB 3190
2400 RETURN
```

The routine looks to see if one side has twice as many units as the other, or if one side has less than two units. A message is displayed to display the outcome.

Other victory conditions could be added. One factor which decided many medieval battles was the death of the leader, but adding this condition to the game could have its own problems. The game then simply becomes centred around one unit.

Another condition might be that half of one army had reached the opponent's baseline. Half could be measured either in units, or in total strengths of units.

Yet another criterion for victory could be the absolute number of casualties caused, in which case a variable containing the total number of casualties would have to be set up to count the dead after each exchange.

LET BATTLE COMMENCE

You now have all the routines that make up Cavendish Field. All you need now to marshal your troops is the main loop.

```
10 CLEAR
30 GOSUB 190
40 GOSUB 470
50 GOSUB 860
60 REM Dummy for Repeat loop
70 FOR i=1 TO 8
80 IF T(i,1)<4 THEN GOSUB 1380: IF y>2
   THEN GOSUB 1900
90 IF T(i,1)<5 THEN INK 1: PRINT AT
   T(i,8),T(i,9);u$(i)
100 NEXT i
110 FOR e=9 TO 16
120 IF T(e,1)<4 THEN GOSUB 2140
130 NEXT e
140 GOSUB 1020
150 GOSUB 2290
160 IF vc<>1 AND de<>1 THEN GOTO
   60
170 GOSUB 2380
180 STOP
2410 REM Delay
2420 PRINT AT 21,7;"{PRESS ANY KEY}"
2425 LET g$=INKEY$: IF g$="" THEN
   GOTO 2425
2430 RETURN
```

```
35 GOSUB 190
40 GOSUB 470
50 GOSUB 860
60 REM DUMMY FOR REPEAT LOOP
70 FOR I=0 TO 7
75 Y=0
80 IF T(I,0)<>4 THEN GOSUB 1380:IF
   Y>2 THEN GOSUB 1900
90 IF T(I,0)<>4 THEN P=T(I,7):Q=T(I,8):
   GH=VAL(MID$(U$,I+1,1))+67:CL=1:
   GOSUB 2600
100 NEXT I
105 GOSUB 2540
110 FOR E=8 TO 15
120 IF T(E,0)<>4 THEN GOSUB 2140
130 NEXT E
140 GOSUB 1020
150 GOSUB 2290
160 IF VC<>1 AND DE<>1 THEN 60
170 GOSUB 2380
180 END
2410 REM DELAY
2420 PRINT "□□□□□□□□□□
   [PRESS ANY KEY]"
2425 GET G$:IF G$=""THEN 2425
2430 RETURN
```

```
10 MODE1
30 PROCinit:PROCcr:PROCds
60 REPEAT
70 FORi=0TO7
80 IF T%(i,0)<>4 THEN PROCunitsel : IF
   yn>2 THEN PROCactsel
90 COLOUR1:IF T%(i,0)<4 THEN PRINTTAB
   (T%(i,7)+1,T%(i,8)+1);MID$(unst$,
   i+1,1)
100 NEXT
110 FORen=8TO15
120 IF T%(en,0)<>4 THEN PROCensl
130 NEXT
140 PROCeffect:PROCvicdef
160 UNTIL vic OR def
170 PROCmess
180 END
2410 DEF PROCpause
2420 PRINT TAB(10,30);"[PRESS ANY
   KEY]":G=GET
2430 ENDPROC
```

```
10 CLEAR 500:PMODE 3,1:COLOR 2,1:PCLS:
   SCREEN1,0:DU=RND(-TIMER)
30 GOSUB 190
40 GOSUB 470:GOSUB 3130
50 GOSUB 860
60 REM
70 FOR I=1 TO 8
80 IF T(I,1)<4 THEN GOSUB 1380:IF Y>2
   THEN GOSUB 1900
90 IF T(I,1)<5 THEN COLOR 3:DRAW
   "BM"+STR$(T(I,9)*8)+","+STR$
   (T(I,8)*8):UU=VAL(MID$(U$,I,1)):A$=
   UC$(UU):GOSUB 3000
100 NEXT I
110 FOR E=9 TO 16
120 IF T(E,1)<4 THEN GOSUB 2140
130 NEXT E
140 GOSUB 1020
150 GOSUB 2290
160 IF VC<>1 AND DE<>1 THEN 60
170 GOSUB 2380
180 GOSUB 2410:CLS:END
190 REM INITIALIZATION
200 VC=0:DE=0
2410 REM DELAY
2420 DRAW"BM80,176":A$="PRESS ANY
   KEY":GOSUB 3190
2425 G$=INKEY$:IF G$="" THEN 2425
2426 LINE(0,176)-(255,183),PRESET,BF
2430 RETURN
```

BBC owners with a disk filing system should type the following before they want to RUN the saved program:

```
*TAPE
FORA%=0TO&1980:?(&E00+A%)=
?(PAGE+A%):NEXT
PAGE=&E00
OLD
```

First, all except the Commodore and Acorns CLEAR some memory space. The Acorn and Dragon/Tandy set up the graphics mode. Routines to initialize the game and draw the map are then called.

The main loop begins at Line 60 and ends at Line 160. It cycles through eight requests for the player to give orders, eight calls to the computer's order selecting routines—one for each unit on the field. It then has 16 calls to the routine that allows the orders to take effect, followed by one call to the test for victory or defeat. Finally, the victory message is printed if the victory test was found true.

At the end of the main loop is a short routine (starting at Line 2410) which is used to pause the game at appropriate moments.

PLAYING INSTRUCTIONS

Immediately you RUN the program the computer gets on with drawing the map. The terrain symbols appear first, followed by the opposing units, and the border.

Now's the time to start building your strategy. A series of prompts appear in the text 'window'. Starting from unit one, the unit number and description—for instance, knights—along with the current orders—halt, perhaps. The player is asked Change (Y/N)?

If the answer is Y, a menu of order options is displayed: Fire, Halt or Move. The Fire option is only open to the archers, so any attempt to make another type of unit fire will make the message 'No Bows' appear, and the machine will wait for another choice. If the Move option is chosen, the prompt 'Which way? (N,S,E,W)' appears, ready for the player's choice.

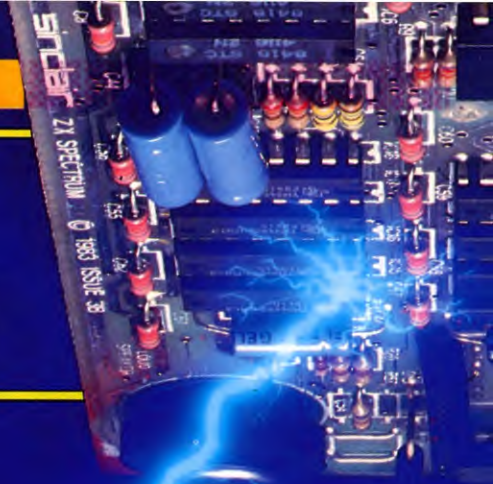
This process is repeated for each unit—the one that's highlighted is the one you have to make the decision on.

THE AFTERMATH

At this stage, Cavendish Field is about as simple as a wargame can be, but still be playable. There are numerous opportunities to tinker with various aspects of the program.

In the final part you will see how the computer can be made into a cleverer opponent, but before you add the new routines, try playing the game as it stands. Beginners, in particular, can get to grips with the strategies involved in wargaming.

WARGAMING: MILITARY INTELLIGENCE



Turn your computer opponent from Ethelred the Unready into Genghis Khan with these additions to Cavendish Field. Stronger strategies result from using heuristics

Your completed wargame probably doesn't offer too much of a challenge at the moment. The program only allows random movement by the computer, so it's quite simple to outwit the machine after you've played a few games. Virtually any simple but sensible strategy will be effective against the computer simply because the computer has its plan, and doesn't respond to the player's tactics.

When you become bored with winning every time, the thing to do is to make the computer's game stronger. This means that you have to program in new routines. But as one of the problems with writing a program like Cavendish Field is coping with memory restrictions—particularly on the Acorn machines—any additions to the program have to be simple but effective.

A STRONGER OPPONENT

Two approaches can be used to make the computer harder to beat. The easy option is to accept the weakness of the computer's tactics and give it stronger forces than the player. This is an approach that has been adopted in some commercial wargames because this is by far the simplest to program.

If you want to see the effect of this simple approach, it can be added to the program with just one or two lines, as follows. In a moment, you'll see how to tackle a more sophisticated answer to the problem. This involves different additions, so you may want to read on first, or delete the 'extra strength' lines after you have tried them.

In the Spectrum, Dragon and Tandy programs, the unit's initial strength is held in element 6 of the troop array, and in the others it's held in element 5. To increase the computer's unit strength you only need a simple addition:

```
665 IF j=8 THEN LET T(j+i,6)=T(j+i,6)+
    FN r(100): LET T(j+i,7)=T(j+i,6)
```

```
655 IF J=8 THEN T(I+J,5)=
    T(I+J,5)+FNR(100)
```

```
650 IF j=8 THEN T%(i+j,5)=T%(i+j,5)+
    RND(100):T%(i+j,6)=T%(i+j,5)
```

```
665 IF J=8 THEN T(J+I,6)=T(J+I,6)+
    RND(100):T(J+I,7)=T(J+I,6)
```

The program adds a random factor to the initial strength and stores it in the current strength element of the troop array.

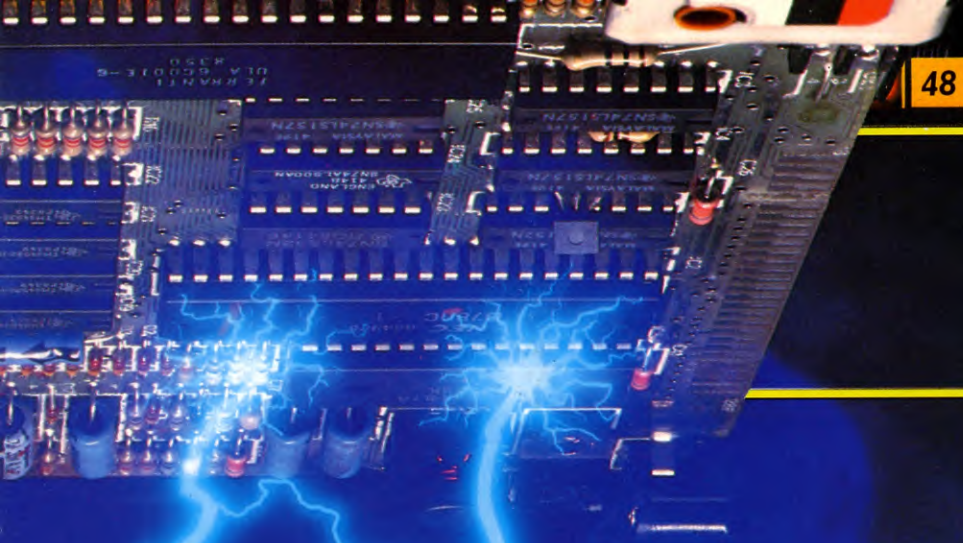
MILITARY INTELLIGENCE

Taking on an opponent embodying brute force and ignorance is not as satisfying as taking on an opponent, with evenly matched forces, that is intelligent. But adding intelligence to the computer's play is a far more difficult problem than adding strength.

The ideas behind adding intelligence to Cavendish Field are somewhat different from those you have seen in Othello (starting on pages 980 to 984) and Fox and Geese (starting on pages 1096 to 1100).

Consider the problem the computer has to solve: in the two board games the play has been very strictly defined. In both there is a clear way forward, having strict ways of judging success, either through tree searching or simpler ways of looking at the board. In both games the program had a strict *algorithm* built in with no uncertainty or random element. In the case of Fox and Geese, at the higher levels, the algorithm is extremely effective and should give most people a strong challenge. In the case of Othello, the algorithm is much simpler, giving the player much less of a challenge.

Returning to the wargame, you'll soon find that an algorithm is almost impossible to construct. There isn't a clearly defined way forward for either side—in fact, it is here that the differences between chess and wargaming become apparent. In chess, there are no random elements, and everything that has to be considered is connected with moves and



spatial relationships on the board. In the case of a wargame, there is almost certainly going to be a random element built in, and there are several different sorts of variables to be considered—armour, morale, movement capabilities, numbers and so on, all play a part in the success or failure of an army. Some variables are interdependent, some are not. And, more importantly, no-one is sure what wins a battle. You cannot say 'If I follow such-and-such a course I will win.'

From a knowledge of how your wargame has been written, and the assumptions made, it could be possible to construct an algorithm, but it would be very, very complex, making the resulting programming large and slow. In practice, the answer to the problem of having a large, unfavourable, or non-existent algorithm is to resort to *heuristics*.

A heuristic is simply a 'rule of thumb' which seems likely to work in some situations. It is important to realise that heuristics are not guaranteed to work, and may even make a disastrous error on some occasions. The rule, though, is generally worth trying. In fact, heuristics are precisely what everyone uses in everyday life because of the complexity of many of the problems we are faced with.

The program, then, should have a list of heuristics built in. They could look like this:

- If you are a unit of archers, and there is an enemy unit within range, then fire.
- If you are involved in a battle, but are losing, then try going the opposite way.
- If you are near an enemy unit which is stronger than you, move away from it.
- If you are near a unit which is weaker than you, attack it.

In the last case, you will see that the computer has an advantage over you since it knows your exact strength, while you do not know its strength. However, it is worth allowing the computer to use this as it allows the machine more power of decision.

AN INTELLIGENT APPROACH

Enough of the theory—the computer needs an overall plan. No battle is fought without some kind of plan of action. The plan will depend on the victory conditions for the game—if you could win by killing the opposing general, then one possible plan would be to send all units against the general's head-

quarters. In Cavendish Field, victory can only be achieved by killing more of the enemy than he kills of you, so the computer wants a plan which will achieve this.

One simple method of trying to destroy enemy troops is to attack weak units with strong forces. This means that a plan saying 'Only attack an opponent who is weaker than you' could be employed. In fact, this is actually quite a complex tactic to program. It can be simplified by saying 'concentrate all your forces on one square'. By doing this (and assuming that the player doesn't concentrate on the same square), the computer's forces should outnumber the player's. A repercussion of all this is that the player's forces will outnumber the computer's elsewhere—this is where the heuristic falls down—so the other, outnumbered forces should avoid combat.

So that the player cannot readily anticipate the computer it must have a number of choices open to it. Again, to keep it simple, you can make the chosen square one occupied by one of the player's units. There is a routine each turn which tests to see if the chosen square is to be changed. This should keep the player guessing as to the computer's exact plan, and keep the computer's forces following the same pattern.

The second aspect of play which must be controlled intelligently is the individual action of each unit. Irrespective of the overall plan, each unit must respond to the immediate situation on the battlefield around it. A unit shouldn't, for example, make straight for the concentration point if many enemy units block its way.

When the computer issues its orders, it makes a series of tests. They are carried out in a carefully considered order, bearing in mind two considerations. Firstly, the tests should be conducted as rapidly as possible. This means that it should be possible to end the test routine the moment a decision has been made, without the need to continue with unnecessary tests. Secondly, the most important tests should come first. If this is not the case, the computer might make a decision according to a less important criterion.

The rules used in the program are as follows, with the most important first:

- If the unit is in combat and won last turn, then continue with the same order. No other rules will be tested if this is true.
- If the unit is in combat, but lost last turn, then it should move away from its opponent. This will not always be possible because the square the unit should move into may be already occupied, or it may be at the map edge.
- If the unit consists of archers and a target is within range, it should fire. Putting the rule here ensures that the archers will always take the less risky option of firing rather than taking part in hand-to-hand combat.
- If there is an enemy unit which is within maximum movement distance, and the unit is weaker than you, attack that unit. If, however, the unit is stronger than you, then retreat. This is a long search because enemy units on the field must be considered.
- Move towards the concentration square.

Even these few rules take a fair time to operate and, if they are all tested for each unit, the game can slow down considerably. Unfortunately, this is the penalty for programming intelligence in BASIC, as you saw in Fox and Geese—but don't worry, there are none of the huge waits you may have experienced in that game.

ADDITIONAL ROUTINES

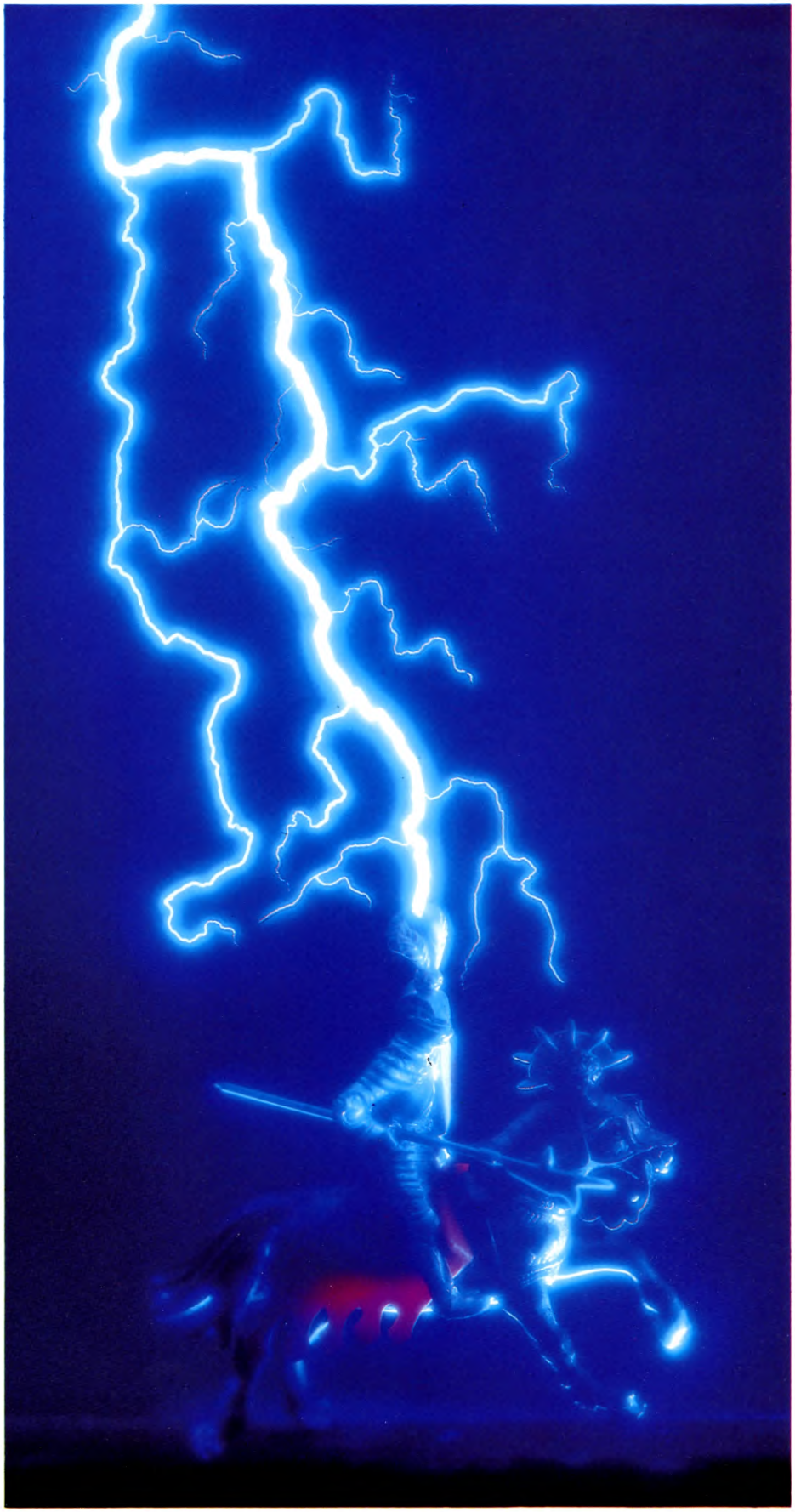
Before adding the new routines you should delete Lines 1770 to 1790 as these now become a separate routine.

Now add these lines:

```
360 DIM t$(8,12): DIM o$(5,12): DIM
    w$(5,9): DIM m$(5,12): DIM a$(4,12):
    DIM r$(4,12): DIM c(8)
416 LET sp=1
1665 IF wn>8 THEN LET c(wn-8)=8
1666 IF lo>8 THEN LET c(lo-8)=T(wn,2)
1760 LET ra=st: LET rb=sh: LET rc=fx: LET
    rd=fy: GOSUB 3000
2140 REM Enemy selects action
2142 REM Dummy for repeat loop
2143 LET r=FN r(10)
2144 IF r=1 OR T(sp,1)>3 THEN LET
    sp=FN r(8)
2145 IF r=1 AND T(sp,1)>3 THEN GOTO
    2142
2150 IF c(e-8)=8 THEN RETURN
2155 IF c(e-8)<>0 THEN LET T(e,1)=3:
    LET T(e,2)=c(e-8): LET c(e-8)=0:
    RETURN
2170 IF T(e,3)=2 THEN LET ra=1: LET
    rb=e: LET rc=5: LET rd=5: GOSUB
    3000: IF gp<>-1 THEN LET T(e,1)=1:
    RETURN
2180 LET T(e,1)=3
2181 LET hp=5: LET vp=5: LET mv=0
2182 FOR v=1 TO 8
2183 LET zp=0: GOSUB 3100
2184 IF zp<>0 THEN GOSUB 3200
2185 NEXT v
2187 IF hp<>5 AND vp<>5 THEN
    RETURN
2188 IF mv<>0 THEN LET T(e,2)=mv:
    RETURN
2189 LET hp=T(e,8)-T(sp,8):
    LET vp=T(e,9)-T(sp,9):
    GOSUB 3200
2190 RETURN
3000 REM Target in range?
3010 LET gp=-1
3020 FOR m=ra TO (ra+7)
3030 LET xx=ABS (T(m,8)-T(rb,8)): LET
    yy=ABS (T(m,9)-T(rb,9))
3040 IF xx<rc AND yy<rd AND T(m,1)<4
    THEN LET rc=xx: LET rd=yy: LET gp=m
3050 NEXT m
3060 RETURN
3100 REM Is a weak or a strong unit nearby?
3110 IF T(v,1)>3 THEN RETURN
3120 LET xx=ABS (T(v,8)-T(e,8)): LET
    yy=ABS (T(v,9)-T(e,9))
3130 IF T(v,7)>=T(e,7) AND xx<5 AND
    yy<5 THEN LET mv=T(v,2)
3140 IF xx<hp AND yy<vp THEN LET
    hp=xx: LET vp=yy: LET zp=1
    : RETURN
3150 RETURN
3200 REM Move towards a predetermined
    square
3210 IF hp>=0 THEN LET lp=1
3220 IF hp<0 THEN LET lp=3: LET
    hp=ABS (hp)
3230 IF vp>=0 AND ABS (vp)>hp THEN
    LET lp=2
3240 IF vp<0 AND ABS (vp)>hp THEN LET
    lp=4: LET vp=ABS (vp)
3250 LET T(e,2)=lp
3260 RETURN
```

```
360 DIM T$(7),O$(4),W$(4),M$(4),A$(3),
    R$(3),CF(8)
415 W$="NNYY":S%=0
1665 IF WN>7 THEN
    CF(WN-8)=8
1666 IF LO>7 THEN
    CF(LO-8)=T(WN,1)
1760 RA=BG:RB=SH:RC=FX:RD=FY:
    GOSUB 3000
2140 REM
2142 REM
2143 R=FNR(10)
2144 IF R=1 OR T(S%,0)=4 THEN
    S%=FNR(8)-1
2145 IF R=1 OR T(S%,0)=4 THEN
    2142
2150 IF CF(E-8)=8 THENRETURN
2160 IF CF(E-8)<>0 THEN T(E,0)=2:
    T(E,1)=CF(E-8):CF(E-8)=0:RETURN
2170 IFT(E,2)=1THENRA=0:RB=E:RC=5:
    RD=5:GOSUB3000:IFGP<>-1THEN
    T(E,0)=0:RETURN
2180 T(E,0)=2
2181 H%=5:V%=5:MV=0
2182 FOR SC=0 TO 7
```

```
2183 Z%=Ø:GOSUB31ØØ
2184 IF Z%<>Ø THEN GOSUB 32ØØ
2185 NEXT SC
2187 IF H%<>5 AND V%<>5 THEN RETURN
2188 IF MV<>Ø THEN T(E,1)=MV:RETURN
2189 H%=T(E,7)-T(S%,7):V%=T(E,8)-T
     (S%,8):GOSUB 32ØØ
219Ø RETURN
3ØØØ REM
3Ø1Ø GP=-1
3Ø2Ø FOR M=RA TO (RA+7)
3Ø3Ø XX=ABS(T(M,7)-T(RB,7)):YY=ABS
     (T(M,8)-T(RB,8))
3Ø4Ø IF XX<RC AND YY<RD AND T(M,Ø)
     <4 THEN RC=XX: RD=YY:GP=M
3Ø5Ø NEXT M
3Ø6Ø RETURN
31ØØ REM
3111Ø IF T(SC,Ø)=4 THEN RETURN
3112Ø XX=ABS(T(SC,7)-T(E,7)):YY=ABS(T
     (SC,8)-T(E,8))
3113Ø IF T(SC,6)>=T(E,6) AND XX<5 AND
     YY<5 THEN MV=T(SC,1)
3114Ø IF XX<H% AND YY<V% THEN H%=
     XX:V%=YY:Z%=1:RETURN
3115Ø RETURN
32ØØ REM
3211Ø IF H%>=Ø THEN L%=2
3122Ø IF H%<Ø THEN L%=4: H%=ABS(H%)
3123Ø IF V%>=Ø AND ABS(V%)>H% THEN
     L%=1
3124Ø IF V%<Ø AND ABS(V%)>H% THEN
     L%=3:V%=ABS(V%)
3125Ø T(E,1)=L%
3126Ø RETURN
```



Retype the original Lines 21Ø to 29Ø plus a new Line 3ØØ CHAIN"GAME". SAVE this as "WARGAME". Now LOAD your existing game. Delete Lines 21Ø to 29Ø and Lines 65 and 2Ø1Ø and then add the following lines (some of which overwrite existing lines). SAVE this after "WARGAME" as "GAME". To use the program, LOAD and RUN "WARGAME", which will then CHAIN the main program.

```
36Ø DIMtype$(7),order$(4),cf(8)
37Ø dir$="NWSE"
38Ø FORi=ØTO4:READorder$(i):NEXT
39Ø FORi=ØTO7:READtype$(i):NEXT
415 S%=Ø
1665 IF win>7 THEN cf(win-8)=8 ELSE
     cf(lose-8)=T%(win,1)
176Ø PROCrng(st,sht,fx,fy)
193Ø FORj=Ø TO 3
198Ø A%=(INSTR("FfHhMm",g$)+1)DIV2-1
214Ø DEF PROCensl
2142 REPEAT
2143 R=RND(1Ø)
2144 IFR=1 OR T%(S%,Ø)=4 THEN
```

```
    S%=RND(8)-1
2145 UNTIL R<>1 OR T%(S%,Ø)<>4
2150 IF cf(en-8)=8 THEN ENDPROC
2155 IF cf(en-8)<>Ø THEN T%(en,Ø)=2:
    T%(en,1)=cf(en-8):cf(en-8)=Ø:
    ENDPROC
2170 IF T%(en,2)=1 THENPROCrng
    (Ø,en,5,5):IF G%<>-1 THEN T%(en,
    Ø)=Ø:ENDPROC
2180 T%(en,Ø)=2
2181 H%=5:V%=5:mv=Ø
2182 FORsc=Ø TO 7:Z%=Ø:PROCwk
2184 IFZ%<>Ø THEN PROCvg(en)
2185 NEXT
2187 IFH%<>5 AND V%<>5 THEN
    ENDPROC
2188 IF mv<>Ø THEN T%(en,1)=mv:PRINT
    "Retreat":ENDPROC
2189 H%=T%(en,7)-T%(S%,7):V%=T%
    (en,8)-T%(S%,8):PROCvg(en)
2190 ENDPROC
2570 DATAfire,halt,move,status,rout,knights,
    sergeants,men-at-arms,men-at-arms,archers,
    archers,peasants,peasants
3000 DEF PROCrng(a,b,c,d)
3005 G%=-1
3010 FOR m=a TO (a+7)
3020 IF ABS(T%(m,7)-T%(b,7))<c AND
    ABS(T%(m,8)-T%(b,8))<d AND T%(m,
    Ø)<4 THEN c=ABS(T%(m,7)-T%(b,7)):
    d=ABS(T%(m,8)-T%(b,8)):G%=m
3030 NEXT
3040 ENDPROC
3100 DEF PROCwk
3105 IF T%(sc,Ø)=4 THEN ENDPROC
3110 IF T%(sc,6)>=T%(en,6) AND (ABS
    (T%(sc,7)-T%(en,7))<5 AND ABS
    (T%(sc,8)-T%(en,8))<5) THEN mv=T%
    (sc,1)
3120 IF ABS(T%(sc,7)-T%(en,7))<H%
    AND ABS(T%(sc,8)-T%(en,8))<V%
    THEN H%=T%(sc,7)-T%(en,7):V%=
    T%(sc,8)-T%(en,8):Z%=1:ENDPROC
3130 ENDPROC
3200 DEF PROCvg(en)
3210 IF H%>=Ø THEN L%=2
3220 IF H%<Ø THEN L%=4:H%=ABS(H%)
3230 IF V%>=Ø AND ABS(V%)>H% THEN
    L%=1
3240 IF V%<Ø AND ABS(V%)>H% THEN
    L%=3:V%=ABS(V%)
3250 T%(en,1)=L%
3260 ENDPROC
```

If you have a DFS then type this after SAVEing the program but before RUNning it.

```
*TAPE
FOR A%=Ø TO &1980:?(&EØØ+A%)=
    ?(PAGE+A%):NEXT
PAGE=&EØØ
OLD
```

Before you type in these lines, the routines starting at Line 3ØØØ need to be moved downwards to make space. Type:

```
RENUM 4ØØØ,3ØØØ
```

Then enter these lines:

```
36Ø DIM T$(8),O$(5),W$(5),M$(5),A$(4),
    R$(4),C(8)
416 SP=1
1665 IF WN>8 THEN C(WN-8)=8
1666 IF LO>8 THEN C(LO-8)=T(WN,2)
1760 RA=ST:RB=SH:RC=FX:RD=FY:
    GOSUB3ØØØ
2140 REM ENEMY SELECTS ACTION
2142 REM DUMMY FOR REPEAT LOOP
2143 R=RND(1Ø)
2144 IF R=1 OR T(SP,1)>3 THEN
    SP=RND(8)
2145 IF R=1 AND T(SP,1)>3 THEN 2142
2150 IF C(E-8)=8 THEN RETURN
2155 IF C(E-8)<>Ø THEN
    T(E,1)=3:T(E,2)=C(E-8):C(E-8)=Ø
    :RETURN
2170 IF T(E,3)=2 THEN RA=1:RB=E:
    RC=5:RD=5:GOSUB3ØØØ:IF
    GP<>-1 THEN T(E,1)=1
    :RETURN
2180 T(E,1)=3
2181 HP=5:VP=5:MV=Ø
2182 FOR V=1 TO 8
2183 ZP=Ø:GOSUB 31ØØ
2184 IF ZP<>Ø THEN GOSUB 32ØØ
2185 NEXT V
2187 IF HP<>5 AND VP<>5 THEN
    RETURN
2188 IF MV<>Ø THEN T(E,2)=MV:
    RETURN
2189 HP=T(E,8)-T(SP,8):VP=T(E,9)-
    T(SP,9):GOSUB32ØØ
2190 RETURN
3000 REM TARGET IN RANGE?
3010 GP=-1
3020 FOR M=RA TO (RA+7)
3030 XX=ABS(T(M,8)-T(RB,8)):YY=ABS
    (T(M,9)-T(RB,9))
3040 IF XX<RC AND YY<RD AND
    T(M,1)<4 THEN
    RC=XX:RD=YY:GP=M
3050 NEXTM
3060 RETURN
3100 REM IS A WEAK OR A STRONG UNIT
    NEARBY?
3110 IF T(V,1)>3 THEN RETURN
3120 XX=ABS(T(V,8)-T(E,8)):YY=ABS(T
    (V,9)-T(E,9))
3130 IF T(V,7)>=T(E,7) AND XX<5 AND
    YY<5 THEN MV=T (V,2)
3140 IF XX<HP AND YY<VP THEN
```

```
    HP=XX:VP=YY:
    ZP=1:RETURN
3150 RETURN
3200 REM MOVE TOWARDS A
    PREDETERMINED SQUARE
3210 IF HP>=0 THEN LP=1
3220 IF HP<0 THEN LP=3:
    HP=ABS(HP)
3230 IF VP>=0 AND ABS(VP)>HP THEN
    LP=2
3240 IF VP<0 AND ABS(VP)>HP THEN
    LP=4:VP=ABS(VP)
3250 T(E,2)=LP
3260 RETURN
```

HOW IT WORKS

The additions before Line 2140 initialize some extra variables and DIMension some new arrays which will be used in the intelligence routines.

At the start of the Select Action routine there is a one in ten chance of the computer changing its target square. The target square is checked and adjusted if necessary, but the computer takes no action on it at this stage—the rest of the tests have to be carried out first.

If the unit consists of archers, Line 2170 uses the In Range routine to decide whether to fire at an enemy unit. If the archers do not fire, or the unit is of any other type, Line 2180 changes the order to move. The next section of program checks each of the enemy units. Line 2183 uses the Weak or Strong Unit subroutine to determine if the computer's unit should move towards (engage in combat) or move away from the nearest unit belonging to the player.

If the unit controlled by the computer isn't engaged in combat, or about to do so, the program uses the Move Towards subroutine (at Line 3500) to move the unit towards the target square.

YOUR CHOICE

You can incorporate either or both of the extra strength or intelligence routines to improve the computer's play at Cavendish Field—according to the strength of opponent you feel competent to take on.

The very nature of heuristics means that the rules of thumb that have been incorporated into the program aren't necessarily the best in all circumstances. And some players will find these tactics easier to outwit than others.

Only trial and error can suggest the best compromises and you may wish to try some different routines—you may well find you can further strengthen the computer's play with a little effort.