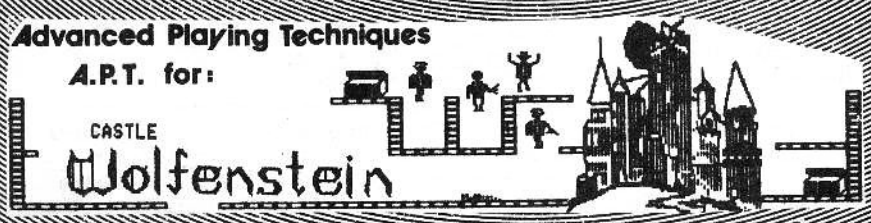
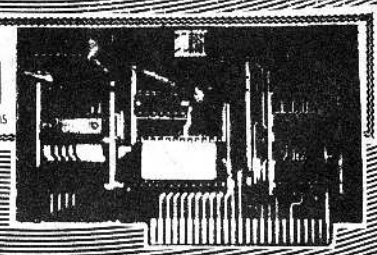
HARD A.P.T. COMPUTING

UPDATE 3.1 DEC 82





NOT COPY-PROTECTED

GraFORTH®

by Paul Lutus

Make Your Graphics Come Alive!

GraFORTH combines sophisticated graphics features with a powerful programming language. Much more than a utility program, GraFORTH's superior graphics make it the ultimate language for entertainment and educational software creation. Included are plotting and line graphics, text display and character image graphics, and high speed 3-D graphics, all with a variety of colors and drawing options. GraFORTH can be used on a 48K Apple][system with DOS 3.3 and one disk drive. A 16K memory card is a useful option.

The Language:

GraFORTH is a graphics language similar in structure to FORTH, but entirely rewritten for ease of use and maximum speed. (Counting to 30,000 in GraFORTH takes less than three seconds!) Immediate commands and programs can be entered and run directly from the keyboard. GraFORTH includes a full set of arithmetic and string handling capabilities. Since standard DOS files are used, communication with other programs and languages is straightforward.

Plotting and Line Graphics:

The first level of graphics consists of plotting points, drawing lines, and filling areas in any of the Apple's high-resolution colors. Lines are drawn much faster than in Basic, and colored lines are never broken. Lines and areas can also be neatly erased from the screen without disturbing other images. Turtle-graphics are included to draw line shapes rapidly at any angle.

Text Display and Character Graphics:

GraFORTH displays both upper and lower case characters. You can use any of the five character sets provided, or create your own with the character editor. Character shapes may be combined to form a single multi-character image, then "block printed" at high speed anywhere on the Apple screen. Characters and character shapes can be drawn in color up to 8 times their normal size.

Three-Dimensional Graphics:

GraFORTH can also draw three-dimensional color images at speeds that make animation possible! Up to sixteen 3-D objects can be manipulated simultaneously. Images can be rotated, scaled, translated, and positioned, with or without perspective. The supplied image editor allows you to create your own 3-D images. Colors may be specified as an image is created, or selected when the image is drawn.

Music:

GraFORTH includes a sophisticated software-based music synthesizer for adding music or sound effects to your programs. Music can be played in any one of several instrument voices.

The System:

Programs written in GraFORTH can be saved to disk as complete stand-alone systems that do not require any additional software to run. This makes GraFORTH the ideal language for developing games and other graphics software.

The Package:

GraFORTH is supplied on a diskette with a special version of DOS 3.3 that loads into a language card (if present), freeing up more memory for your programs. The disk includes many sample image files, utilities, and complete demonstration programs detailing the features of GraFORTH. Included with the disk is a 200-plus page tutorial explaining the ins and outs of GraFORTH. No previous programming experience is necessary to use GraFORTH.

Order GraFORTH from Computer Hideout today!

COMPUTER HIDEOUT

INTELLIGENT PRODUCTS FOR MICROCOMPUTERS

$64.95

TERMS: We Accept M.C. or VISA (Indicate Card # and Expiration Date) Personal Checks - Allow 3 weeks to clear. Please Enclose $2.00 for Postage and Handling. Louisiana Residents - Please Add 6% Sales Tax.

(318) 942-9446

P. O. Box 264 / Opelousas, Louisiana 70570

Charles R. Haight
Publisher &
Editor-in-chief

Bev R. Haight
Associate Editor

Julie Joringdal
Editorial Assistant

Beryl Flynn
Subscriptions &
Distribution

Karen Fitzpatrick
Comptroller

Contibuting writers

Bobby
Julie Joringdal
Mycroft
Robb Canfield

Published by
Softkey Publishing
P.O. Box 44549
Tacoma, WA 98444
(206) 581-6038

Printed by
Peninsula Gateway

No responsibility can be assumed for unsolicited manuscripts. We suggest you send only copies. Return postage must accompany all submissions if they are to be returned. Send a self-addressed, stamped envelope for all inquiries that need a reply.

IMPORTANT ANNOUNCEMENTS

Due to a shortage of people and computers...

Starting with issue #4, the HARDCORE Program Library will be selling programs by the issue or volume instead of by individual program.

This means that, like NIBBLE, each issue will have its own disk of programs.

In this way, we can maintain a low overhead and keep the prices to our readers low.

So this is the last chance for you to order individual programs.

The volumes will be:

Disk control..... $18.00
(DiskEdit & DiskView)

Issue #2..... $18.00
(Artist's Easel, Amber's Ts, Text Invaders, Relief Mapper)

Issue #3..... $18.00
(Map Editor, Zyphyr Wars, Menu, I.O.B., HyperDOS)

Subscriptions

U.S.A.	$20
Canada	$29
Mexico	$32
S. America	$38
all others	$42

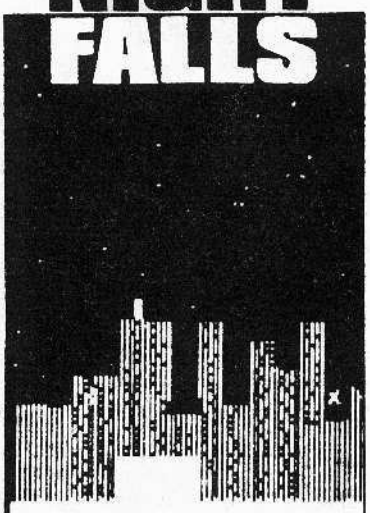
All foreign subscriptions must be in U.S. funds and prepaid. No credit cards, yet.

DOMESTIC DEALER RATES sent upon request, or call (206) 531-1684.

UPDATES will focus on the subject of Softkeys, "How to make backup copies of so-called uncopyable diskettes." Updates are available only to subscribers and are not sold in stores.

If HardCore is not published on schedule, your subscription will still be for 4 issues and 4 updates regardless of the time it takes to fulfill that order.

ADVERTISER'S INDEX



how to use the I.O.B. DATA STATEMENTS

By Bobby

Requirements:
- Back-up copy of the Ultima disk
- A copy of MUFFIN
- One blank disk
- DiskView or other nibbler

A number of subscribers have written asking for examples on how to use data statements in the IOB program. Here is a step-by-step procedure using the program "Ultima".

1. Make a chart similiar to the one shown below. Allow room for tracks 3 to 34.

figure 1

		ADDR			DATA		
TRK	A1	A2	A3	D1	D2	D3
3	DD	AA	F7	DF	AA	FD
4						
5						
etc						

2. RUN DiskView. Insert the Ultima disk then READ track 3 and locate the address and data marks. These can be found by looking for large groups of identical bytes. (see figure 2) When you have found the correct bytes fill in the chart for track 3.

figure 2

```
FF FF FF FF FF FF FF FF FF FF FF FF FF
FF [DD AA F7] FF FE AB AB AF AB FB FE
FE FE FE FE FE FE FE FE FE FE FE FE FE
FE FE FE [DF AA FD] AE DA EF DA FD BE
AE AF FF BF EF DB DB AB EF ED AE EB
```

3. Repeat step 2 for each track from 3 to 34.

4. The normal address and data marks used by DOS are:
   D5 AA 96 / D5 AA AD 16-sector
   D5 AA B5 / D5 AA AD 13-sector
Read track 0 and compare these marks with what you find on the disk to determine whether the disk is 13 or 16 sector. You should find that the disk is 13-sector.

5. Compare the 13-sector marks with the bytes in your chart. Address mark A2 and data mark D2 are the same as their 13-sector counterparts. This means that we can delete them from our chart. That leaves two address marks and two data marks.

6. Use the Byte Conversion Chart in Update 2.1 (or a hex/dec converter) to convert the byte values in your chart into decimal numbers.

7. LOAD the IOB program and enter the numbers into data statements starting with line number 63050. Maintain the order of A1,A3,D1,D3 as you enter the decimal numbers.

8. Enter or change the following lines:

```
   ››› Set buffer start to $3000 ‹‹‹

50  POKE BUF,48 : POKE CMD,CD : POKE
    TRK,TK : POKE SCT,ST : POKE DRV,
    DV : POKE VOL,VL : RETURN

      ››› Read the DATA ‹‹‹

60  READ A1,A2,D1,D2

     ››› POKE DATA into RWTS ‹‹‹

70  POKE 6774,A1 : POKE 6795,A2 :
    POKE 6664,D1 : POKE 6685,D2 :
    RETURN

 ››› Set 13-sector and no chksum ‹‹‹

110  DOS = 12 : POKE 6755,24

      ››› INIT target disk ‹‹‹

120  DV = 2 : CD = IN : GOSUB 50

     ››› Copy only 32 tracks ‹‹‹

1000  FOR TK = 3 TO 34

  ››› Read with 13-sector RWTS ‹‹‹

1010  DV = 1 : CD = RD : IO = 797 :
      GOSUB 50 : GOSUB 60 : GOSUB 80

       ››› Check for VTOC ‹‹‹

1015  IF TK = 17 THEN POKE 12289,20
      : POKE 12290,4

  ››› Write with 16-sector RWTS ‹‹‹

1020  DV = 2 : CD = WR : IO = 768
      : GOSUB 50 : GOSUB 80

        ››› ML subroutine ‹‹‹

63000  FOR X = 768 TO 813 : READ A
       : POKE X,A : NEXT
63010  DATA 169,3,160,8,32,217,3,96,1
       ,96,1,0,0,0,25,3,0,32,0,0,1,0,
       0,96,1,0,1,239,216,169,3,60,8,
       8,120,32,0,30,176,3,40,24,96,
       40,56,96

     ››› BLOAD 13-sector RWTS ‹‹‹

63035  PRINT  CHR$ (4) "BLOAD RW"

  ››› Address and Data mark info ‹‹‹

63050  DATA 221,247,223,253,223,245,
       223,189,221,245,221,253,223,
       247,221,237
63060  DATA 223,183,221,253,223,245,
       223,253,223,245,221,237,223,
       183,223,253
63070  DATA 223,247,221,253,223,247,
       223,253,223,245,221,253,223,
       245,215,253
63080  DATA 223,247,223,253,221,245,
       223,237,223,183,215,253,223,
       245,223,237
63090  DATA 221,247,223,237,221,247,
       223,189,223,247,215,237,221,
       247,215,253
63100  DATA 221,247,215,237,223,247,
       223,253,223,245,215,237,223,
       247,215,253
63110  DATA 223,247,221,253,223,247,
       223,237,223,245,215,253,221,
       245,223,253
63120  DATA 223,247,215,253,223,247,
       223,253,221,247,221,237,223,
       183,223,253
```

9. Delete line numbers 85, 90, and 140. They are not required for our purposes.

10. SAVE This version of IOB under a different name. (IE. COPY ULTIMA)

11. BLOAD MUFFIN from your master disk.

12. BSAVE the 13 sector RWTS image for use by the IOB.

```
BSAVE RW, A$1900, L$800
```

The changes and additions to the IOB are to customize it for this application. Let's take a look at what we have done.

Line 50 moves the buffer up to allow room for the 13-sector RWTS.

Continued On Page 15

modifying IOB for single disk drive

By Robb Canfield

REQUIREMENTS:

48K AII+ with Applesoft in ROM
IOB program from issue 3

One of the simplest ways to unlock a "protected" diskette is to use the Input/Output Block. However, the IOB program requires two disk drives and some people have only one.

Here is a modification for IOB that requires only one disk drive.

IOB normally reads one track at a time into memory and then copies it to the target disk. A single drive copy would entail exchanging disks 34 times before a copy was done (once for each track). This modification will allow more tracks to be read into memory at one time and reduce the number of disk swaps.

Either or both changes can be made to the IOB program. They are not dependant upon each other.

CONVERTING IOB FOR SINGLE DISK DRIVE

1) Change the drive number in line 120 to 1.

```
120 VL = PEEK (OVL):DV = 1:CD = IN:
    GOSUB 50
```

2) Change the prompt at line 130.

```
130 A$ = "INSERT BLANK DISK IN DRIVE
    1.": GOSUB 40: CALL IO
```

3) Change whichever control routine you are using to prompt for disk swapping. Lines 1002 and 1012 were chosen because they did not interfere with some of the mods made to the controller (such as mods for unlocking Castle Wolfenstein).

```
1002 A$ = "INSERT SOURCE DISK": VTAB
     12: GOSUB 30: GOSUB 40

1012 A$ = "INSERT TARGET DISK": VTAB
     12: GOSUB 30: GOSUB 40
```

4) Line 1020 in the controller must be changed so that drive one is called.

```
1020 DV = 1: CD = WR: GOSUB 50: GOSUB
     90: GOSUB 80
```

(The actual format of line 1020 may change, but always make DV = 1 if you use a single disk drive)

5) SAVE this new version.

```
SAVE IOB.SINGLE
```

READING MORE TRACKS WITH IOB

1) Change the LOMEM in line 10 to give us more room.

```
10  TEXT: HOME: LOMEM:37120: GOSUB
    63000: GOTO 100
```

The next step depends on which controller you are using:

If the controller has a loop from 0 to 34, use CHANGE 1.

If your loop goes from 3 to 34, use CHANGE 2.

The changes reflect the ability to divide the loop evenly. The maximum number of tracks which can be read into memory is 7 (ie. 3 to 34 is 32 tracks and the largest number that divides evenly into 32 that is 7 or less is 4, so we use a loop of four tracks.)

CHANGE 1

1) Change the loop variable in line 1000.

```
1000  FOR LO = 0 TO 34 STEP 7
```

2) Change line 80 to read 7 tracks at a time.

```
80  FOR TK = LO TO LO + 6: POKE
    TRK,TK: FOR ST = 0 TO DOS: POKE
    SCT,ST: GOSUB 20: CALL IO: POKE
    BUF, PEEK (BUF) + 1: NEXT : NEXT
    : RETURN
```

3) Change line 85 to also read 7 tracks.

```
85  FOR TK = LO TO LO + 6: POKE TRK,
    TK: FOR ST = 0 TO DOS * 2 STEP 2
    : POKE SCT,ST: GOSUB 20: CALL IO:
    POKE BUF, PEEK (BUF) + 1: NEXT :
    NEXT : RETURN
```

4) Save your new IOB.

```
SAVE IOB+7
```

CHANGE 2

1) Change the loop variable in line 1000.

```
1000 FOR LO = 3 TO 34 STEP 4
```

2) Change line 80 to read 4 tracks at a time.

```
80  FOR TK = LO TO LO + 3: POKE TRK,
    TK: FOR ST = 0 TO DOS: POKE SCT,
    ST: GOSUB 20: CALL IO: POKE BUF,
    PEEK (BUF) + 1: NEXT : NEXT :
    RETURN
```

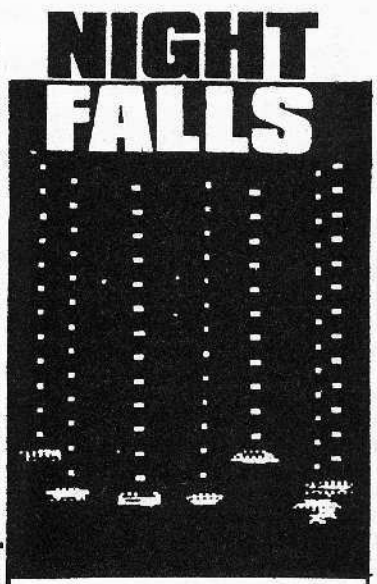
3) Change line 85 to also read 4 tracks.

```
85  FOR TK = LO TO LO + 3: POKE TRK,
    TK: FOR ST = 0 TO DOS * 2 STEP 2
    : POKE SCT,ST: GOSUB 20: CALL IO:
    POKE BUF, PEEK (BUF) + 1: NEXT :
    NEXT : RETURN
```

4) Save your new IOB.

```
SAVE IOB+4
```

UFOs drip from the stars as soon as...



When night falls, the alien invaders drip down from the stars, oozing back up into the sky when day returns. Some, depleted of energy, ooze away in the midst of battle. Each UFO is different in shape and color from any of the others.

bootcode tr

By Mycroft

Requirements:
- Knowledge of machine language
- 48K AII with Integer card
- Multi-Disk Catalog by Sensible Software
- Initialized blank disk with HELLO program deleted.

If you have a little knowledge of machine language programming, and a good measure of perseverance, this article will describe a method of defeating the locking scheme used in a large group of programs and let you capture them on standard DOS. These are the kinds of programs that boot and run with no subsequent disk access. Most games and many utility and business programs fall into this category.

THE THEORY

The basis for this approach is: NO MATTER WHAT LOCKING SCHEME IS USED THE PROGRAM MUST BOOT ON A STANDARD APPLE IN ORDER TO RUN.

If we could somehow step through the Boot process, get everything loaded, and then stop just prior to going to the start of the program, we would be able to save the whole thing and run it under standard DOS.

The APPLE Boot process starts with Boot 0 in the Disk Controller ROM. This short machine language program BLOADS track 0, sector 0 (containing Boot 1 of the disk being booted) at locations $800 thru $8FF, and then jumps to $801 to execute Boot 1.

Boot 1 reads Boot 2, and the process continues through successive Boot stages until finally the main program is loaded and run.

Assuming your disk controller card is in slot 6, the code for Boot 0 starts at $C600 and extends through $C6FA. If you power up your APPLE, enter the monitor and do a "C600L", the beginning of the disassembly listing of Boot 0 can be seen. If you type a few more L's, you will be able to see the jump to Boot 1 at $C6F8.

What we need to do is execute Boot 0, but stop before jumping to Boot 1.

How is this done?

RESET

The RESET routine is at $FF59 in the monitor ROM. It performs a function similar to pushing the RESET button. If called, a RESET cycle is performed and any executing program will be stopped.

If we could somehow get Boot 0 to jump to $FF59 instead of $801, we will have accomplished our first objective.

MODIFYING BOOT 0

But since Boot 0 is in ROM, it cannot be directly modified. The solution is to move it to RAM, using the monitor's Memory Move routine, so we can change it to suit our needs.

The new location should be somewhere in RAM where it will not be overwritten in any of the successive Boot stages.

Memory that is just below DOS is usually safe to use, since many locked programs use only slightly modified versions of normal DOS.

Because of the routine in Boot 0 that finds the slot the Disk Controller is in, the second digit of the address we move Boot 0 to must be the same as the slot number.

Let's use $9600-$96FF.

THE FIRST STEP

From the monitor, type:

```
9600<C600.C6FFM
```

to move Boot 0 and, instead of jumping to $801 at the end, have it jump to RESET by entering:

```
96F8:4C 59 FF
```

Remember to press the RETURN key after each line is entered.

Now put a program disk in drive 1 and execute the modified Boot 0 by typing:

```
9600G
```

The drive will start, and in a second or two you should hear a beep and see the monitor prompt on the screen. But the drive is still spinning, you say? You can turn it off by typing:

```
C0E8
```

Now you can have a look at Boot 1 by listing from $801 to see what it does.

```
801L
```

This process can be repeated for each successive stage in the Boot process. Whenever there is a jump out to a new code section, put in good old 4C 59 FF (JMP to RESET) to stop it there. You can examine this code at your leisure.

Don't worry too much about what the code is doing at this point. Look mainly for JMP's out.

A PRACTICAL EXAMPLE

Let's walk through an actual example, from start to finish, of unlocking a protected program using this method.

To illustrate the procedure, I will use the program "Multi-Disk Catalog III", by Sensible Software. Even though this program has been around for a while, it is an excellent and very useful utility (which I recommend that you purchase and add to your library.)

I have examined dozens of other programs which use a virtually identical boot sequence, and they can be unlocked using this same technique with only a few changes (try one of your own once-load programs if you don't have Multi-Disk Catalog).

Have an initialized slave diskette ready and make sure there is a write-protect tab on the disk you are trying to unlock.

UNLOCKING MULTI-DISK CATALOG III

(BOOT 0)

Turn on your APPLE with the disk drive empty, and push RESET to stop the drive. This will keep the APPLE's memory clear.

Now insert the locked disk and get into the monitor with a CALL-151.

Type:

```
9600<C600.C6FFM
```

to move Boot 0. Then fix the JMP out by typing:

```
96FA:98
```

Cracking Revisited

This changes the JMP out to $9801 where, in the next step, we will be moving Boot 1.
At $9801 type:

```
9801:4C 59 FF
```

so we can stop the Boot.
Now type:

```
9600G
```

After the beep, type:

```
C0E8
```

to turn off the drive.

BOOT 1

Look at Boot 1 by typing:

```
801L
```

This code relocates itself to memory page two, loads Boot 2, and finally jumps out to $301 at $841.
Move this code (so it can be modified) by typing:

```
9800<800.8FFM
```

Fix the jump out to go to $9301 by typing:

```
9843:93
```

Stop it at that point with:*

```
9301:4C 59 FF
```

One other byte in Boot 1 must be changed so that our modified code is executed properly, and that is done as follows:

```
9805:98
```

Now do a "9600G" once again, and type "C0E8" to stop the drive.

BOOT 2

The next stage of the Boot normally starts at $301. Move this code by typing:

```
9300<300.3FFM
```

Take a look at the disassembled listing of the code beginning at $9301.
The JMP out of this stage can be seen at $9343, but it is disguised by being an indirect JMP through page 0 location $3E.
If you examine the code in this Boot stage beginning at $931F, you will find that this indirect jump is used repeatedly to go to $25D, but ultimately the indirect jump address is changed to go to the next Boot stage. This change occurs at $933A-$9341.
The final jump is determined by the byte stored in memory location $3CC, which the program increments by 1 before executing the final indirect jump.
If you look at byte $3CC

```
$3CC
```

you will find that it contains value $36 (other programs commonly use $B6 here). This is the high byte of the jump-to address (the low byte has value $00).
The program increments this value by 1, so the final JMP address is to $3700.
What we want to do is let the Boot do all the jumps when it is going to $25D, but stop it before it makes the final jump out to $3700.

HANDLING THE INDIRECT JUMP

We can write a short subroutine to handle the indirect jump for us.
Because zero page location $3E contains the value $5D for all the indirect jumps except the final one, our subroutine merely needs to see if this value changes and, if so, stop the boot. Enter:

```
9000:A9 5D C5 3E D0 03 4C 5D
9008:02 4C 59 4C 59 FF
```

The source code for this routine would look like:

```
9000-A9 5D     LDA #$5D  Load value.
9002-C5 3E     CMP $3E   Same?
9004-D0 03     BNE $9009 No, go RESET
9006-4C 5D 02  JMP $025D Yes, go on.
9009-4C 59 FF  JMP $FF59 Jump RESET.
```

Change the boot code to jump to this subroutine by typing:

```
9343:4C 00 90
```

and do another:

```
9600G
```

BOOT 3

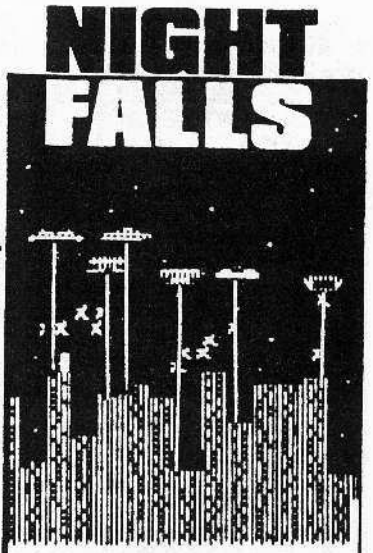
At the beep, you can stop the drive (C0E8), and examine the code beginning at $3700 looking for the next jump out. It is at $3747, and is a jump to $1B03.
So, as before, change this jump by typing:

```
3747:4C 59 FF
```

WRITING TO THE ROM

Now for the sneaky part. The code we moved to memory page $93 ($9300) was responsible for reading this portion of the Boot. But since we just made a change at $3747, we don't want it to be overwritten when we start the Boot over again.
What we do is change byte $93CC, so that a dummy write is done by letting it "write" to the ROM!

Death Rays & X-bombs always follows...



X-bombs descend upon the city, destroying the buildings. They do more damage by intercepting your gravity cannons rays, delaying the the destruction of the UFOs, letting them use their deathrays which atomize the buildings they strike.

This is accomplished by typing:

```
93CC:D0
```

and also changing bytes $9315 and $933E to reference this location instead of $3CC.
Type:

```
9315:93
933E:93
```

to accomplish this feat.
The "write" of the next Boot stage therefore begins at $D000, and is ineffectual except to keep the drive running and in the proper read mode.
Change the subroutine we put in at $9000 to go to the modified next stage by typing:

```
9009:4C 00 37
```

Now type:

```
9600G
```

one last time.
This time, when you hear the beep, the drive will stop by itself. You're almost finished.
Start listing the program at $1B03, looking for the next major jump out.
You should find it at $1C25, and it is a jump to $1E54. Type:

```
1C25:4C 59 FF
1B03G
```

Now list beginning at $1E54. There is an immediate jump to $9D84. List from $9D84.

LANGUAGE CARD?

At $9DE4 and $9DE7 are two indirect jumps, through $9D5E and $9D5C respectively.
If you were to examine the code carefully, beginning at $9D84, you would find that the first indirect jump is taken by systems equipped with language cards (RAM cards), and the second for those without.
No matter, the second indirect address will ultimately be jumped to whichever system you have. To find out what it is, type:

```
9DE7:4C 59 FF
9D84G
```

When you hear the beep, type:

```
9D5C.9D5D
```

and the screen will display (low byte first) the address indirectly jumped to as $33D5. Begin listing from $33D5, and you should find the next JMP way down at $34BC. It goes to $00FD. Once again, type:

```
34BC:4C 59 FF
33D5G
```

The disk drive will start, and the last segment of the program will be loaded. If everything worked correctly, you should hear a beep, the drive will stop, and the screen will be filled with garbage.

WHERE THE PROGRAM STARTS

Normally, the program would next jump via the page zero location we just changed at $34BC to the start of the multi-disk catalog main program. Type:

```
00FDL
```

to see where the start is.
To make it difficult, the software protectors have put one last obstacle in our path. $00FD takes an indirect jump through page zero locations $4E and $4F to the start of the program. Unfortunately, we can't just examine these locations to find out where the jump goes, because they get changed when a RESET cycle is executed.

Not to worry, though, because if you type:

```
348FL
```

you can see that locations $4E and $4F are set from $33C0 and $33C1, respectively. Typing:

```
33C0.33C1
```

will thus divulge (low byte first) the starting address of the main program as $1294.

The program occupies memory from $800-$18FF, $5000-$5CFF, and $9D00-$BFFF. (You find this out by scrolling through memory to try and identify program statements and data, often a trial and error process. If you get too much, no real harm is done, but too little and the program will not run.)
All that remains to be done is to capture the program under normal DOS, and the unlocking process will be complete. If you have managed to follow all this so far, the rest will be easy.

MOVING THE MEMORY

Warm booting a slave diskette will over-write memory locations $800-$8FF and $9600-$BFFF, but everything from $900-$95FF will be unaffected. Let's move the "lower" part of the program ($800-$1800) up and out of the way of the Boot, and put it adjacent to the "middle" part. Do this by typing:

```
3F00<800.18FFM
```

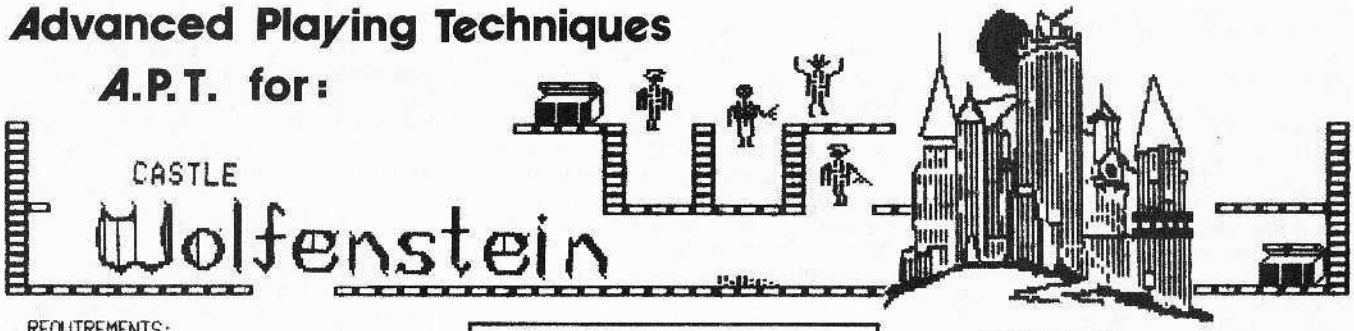
Similarly, move the "top" part of the program down by typing:

```
5D00<800.BFFFM
```

Continued On Page 15

Advanced Playing Techniques

A.P.T. for:



REQUIREMENTS:

The IOB program (from Issue 3 or IOB+ discussed elsewhere in this Update)
MUFFIN from the DOS master disk
Castle Wolfenstein by MUSE
Two disk drives are required unless using IOB+.

INSIDE CASTLE WOLFENSTEIN

Castle Wolfenstein is an arcade/adventure game using HiRes graphics. You are an escaped prisoner of war, trapped in a castle full of Nazi guards and SS troops. You must find the path to freedom and maybe a set of War Plans that are also in the castle.

The game is enjoyable and very addictive. Unfortunately it has one rather annoying routine - every time you run into a wall the screen flickers and a horrible noise issues forth from the speaker.

After playing quite a few games, I became frustrated by this sound and resolved to eliminate it. The first problem that I encountered was our old enemy, software protection.

THE LOCK

Castle Wolfenstein is on a modified 13-sector disk that will boot on 13- or 16-sector APPLEs. The only protection used is to write even sector numbers to the disk. This means that the sectors step by two (ie. 0,2,4,6,8,10). We could solve the first problem (13-sector DOS) with Muffin. Muffin is a program that transfers 13-sector disks to 16-sector. However, Muffin will not handle even numbered sectors. So, I decided to use the IOB program, with a little help from Muffin.

Within Muffin is an image of the 13-sector RWTS. We will use this image when we make our copy. The IOB program must be modified so that the 13-sector DOS in Muffin is called when we read the Castle Wolfenstein disk, and the normal 16-sector DOS is called when we want to write to our copy disk.

Some elements of the illustration for "A.P.T. for Castle Wolfenstein" are from parts of its screen images.

By Robb Canfield

THE KEY

1) Turn on your Apple and boot from the DOS Master disk.

2) Put a blank disk in drive 2 and initialize the blank disk with the name "^HELLO":

```
INIT ^HELLO, D2
```

3) Bload the Muffin program:

```
BLOAD MUFFIN.D1
```

4) Enter the monitor and add a short routine that sets up the RWTS call:

```
CALL -151
18F0:08 78 20 00 1E B0 03 28
18F8:18 60 28 38 60
```

5) Return to Applesoft:

```
3D0G
```

6) Load the IOB program or type it in. This will overwrite the first part of Muffin but does not affect the RWTS part.

7) Change the IOB program to call the 13-sector RWTS in Muffin by adding this line:

```
63015 DATA 169,3,160,8,32,0,16,96
```

8) Change the loop in line 63000 so that our new routine is poked into memory along with the regular data:

```
63000 FOR X = 768 TO 804: READ A :
      POKE X,A : NEXT
```

9) Modify controller #1 so that the appropriate DOS is called:

```
1000 FOR TK=3 TO 34
1005 IO=797 :REM READ FROM A 13
     SECTOR DISK.
1010 DV=1:CD=RD:GOSUB 50:GOSUB 85
1015 IO=768 :REM WRITE TO A 16
     SECTOR DISK.
1020 DV=2:CD=WR:GOSUB 50:GOSUB 80
1030 NEXT
```

10) Change the DOS variable in line 63035 so that only 13 sectors are read and written to the disk, instead of the normal 16:

```
63035 DOS=12
```

11) Delete line 120, (we don't want to initialize the disk) and change line 130 to:

```
130 A$="INSERT BLANK DISK IN
    DRIVE 2.":GOSUB 40
```

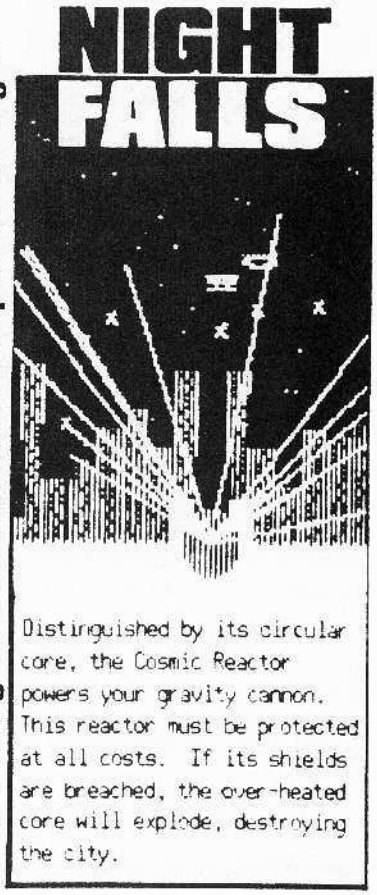
12) Save this version of IOB (so that you won't have to retype it):

```
SAVE IOB-CASTLE
```

13) Insert your copy of CASTLE WOLFENSTEIN in drive 1 (check that the initialized disk is still in drive 2), and RUN the program.

MODIFICATIONS TO CASTLE WOLFENSTEIN

I located three different sound routines. The locations to change are listed in figure 1 in the form of



pokes. To turn these routines ON or OFF follow these steps:

1. BLOAD @WOLF - file to change
2. POKE XXXX,YY - desired change
3. BSAVE @WOLF - save the file

Figure 1

Sound Routine	POKE	ON	OFF
Grenades	4405	48	16
Gun Fire	4045	48	16
Wall Collision	4086	1	0
Screen Flicker	2843	141	96

After making all your changes, return to BASIC (3D0G). Save the program by typing:

BSAVE @WOLF,A$810,L$16EB

Turning off these routines has no effect on the game other than eliminating the specified sound. For example, turning off the wall noise does not turn off the screen flickering, nor does it stop the NAZI's from moving.

STRATEGIES

Any game has it's DOs and DON'Ts, and Castle Wolfenstein is no different. When you are on the first level (a single room with no doors, only a stairway), wait for the guards to be in a prime position before shooting. The guards will not attack you unless you move, attack them, or they bump into you. When you leave a room, try to point the gun in a direction that gives the best chance of scoring a hit. This usually means pointing the gun in the direction you are moving. When entering a new room leave it immediately. This allows you to think about the situation and ready your gun appropriately.

WHEN TO KILL

Try to kill the guards when they are next to a doorway. This stops the other guards and SS from getting to you. The NAZIS's won't cross over fallen bodies. They can still fire at you, they just can't catch you. This can be used to create a safe place from which to throw grenades and such.

SHOOTING THROUGH WALLS

Another handy technique is to shoot through walls. For some reason Castle Wolfenstein will let us fire through corners. This allows us to shoot a guard and not risk being caught. One can also open up chests that are located in a corner. This saves time and avoids unnecessary risks.

ADVANCED PLAYING TECHNIQUES

The following techniques may be considered cheating by the less enlightened, but more open-minded individuals will readily see we are only taking advantage of the program and its limitations.

LIFE BEYOND DEATH

Normally when the RESET key is pressed Castle Wolfenstein saves the current game. We can change this so that instead of saving the game, we exit back to Applesoft. Once there, we can re-boot and resume the game one room back. There is a reset routine in both @INIT and @WOLF, but we are only concerned with the routine in @WOLF for the moment. I have listed the routine here so that it can be easily modified. A complete explanation of how the reset vector works can be found on pages 36 and 37 of the APPLE II Reference Manual.

```
1187-A9 C7    LDA #$C7  Set the
1189-8D F2 03 STA $03F2 reset vector
118C-A9 1E    LDA #$1E  so that it
118E-8D F3 03 STA $03F3 jumps to 1EC7
1191-49 A5    EOR #$A5  Set the power
1193-8D F4 03 STA $03F4 up byte
```

I wanted the APPLE to re-boot the disk when I pressed the RESET key. To

do this, type:

BLOAD @WOLF

Enter the monitor and type the following line:

1191:EA EA

Return to Applesoft (3D0G) and save @WOLF:

BSAVE @WOLF,A$810,L$16EB

NOTE: The ESC key (which saves the game) will still operate normally.

FOR THE AGGRESSIVE PLAYER

Castle Wolfenstein was written so that every room is stored on a unique sector. When the game first starts the track/sector list of CASTLE is read and stored in memory. The first sector contains the variables. Every time you enter a new room, the old room is saved and the new one is read in. This means that any room modifications you have made (grenades are handy for this) will be saved. It also allows us to go back one room if we just happen to make a fatal error. Also, if you have a disk editing program, such as DISKEDIT, you can give yourself 255 bullets and grenades.

THE RIGHT SECTOR

First we have to find where this sector is. Read track $11, sector $C. This is the first sector of the catalog for Castle Wolfenstein. Look for the program ^BACKUP (some copies may have the name BACKUP). If it is not on this sector, try sector $0B of the same track. If you still haven't found it, you are doing something wrong.

After locating the name, back up 3 bytes to find the Track/Sector list. (how this information is stored is explained on pages 129-131 of the DOS Manual). Read this sector. Look at the 13th byte ($0C). The first number is the track where the first sector of the program is located, the next byte is the sector. Read this track/sector. You now have the first sector of the program in memory ready to be modified.

I found the file name ^BACKUP on track $11, sector $0B. The third byte back from the name was $14 and the second byte was $0C, so I read track $14, sector $0C. After reading this sector, I looked at the 13th ($0C) byte. It was $20 and the next byte was $0B. This meant that the first sector of the program was located on track $20, sector $0B.

SOME CUSTOM CHANGES

Once the sector is in memory, move to the proper location and change the byte to the desired value (use the 0 command to move the cursor if you are using DISKEDIT). The table in figure 2 shows the item, the location in the sector and the value to place there. All values are in HEX.

IE. To get 255 bullets, move the prompt to location $47 and change the value there to $FF.

Figure 2

Desired Item	Location	Value
Bullets	$47	0-$FF
Grenades	$48	0-$FF
Uniform	$49	$01
Bullet-Proof Vest	$4A	$01
War plans	$6C	$01
Rank	$6D	see text
Room #	$40	see text
Resurrection	$6F	$00
% Chance of Hit	$4B	0-$FF
% Chance of Recog.	$4D	0-$FF

NOTE: Change both ^BACKUP and ^CASTLE (also known as CASTLE) to be sure the modifications stick. These two files are used alternately at different levels of play.

WHICH ROOM ARE YOU IN?

The map in figure 3 shows the layout of the castle. Each room has a number. This is the number to use if you need to change rooms.

NOTE: You may end up in a wall if you play with the room number. If this happens, you will have to try another room or change your position in the room. Bytes $43-$45 have something to do with your position within the room.

Your rank can be changed to a higher level which will cause the game to be much harder and more interesting. Change the rank byte as follows:

$10 Private	$90 Captain
$30 Corporal	$A0 Colonel
$50 Sergeant	$C0 General
$70 Lieutenant	$E0 Field Marshal

RESURRECTION

If you happen to press RESET too late, your game can still be retrieved if you stop it before playing again. Put a $00 in byte $6F in the sector.

GIVING IT YOUR BEST SHOT

Byte $4B determines the percent chance of your achieving a kill. The higher the number ($FF is greatest), the better your chances.

I'M NOT REALLY HERE

Byte $4D determines the percent chance of your being caught with $FF being the greatest percent chance of being recognized.

SOME MINOR GLITCHES

1) When you have more than 10 bullets, the display will still show you as having only ten bullets. This value will decrement once for each shot fired. Do not get bullets from a box. If you do, the program will replace the actual number of bullets you have with 10.

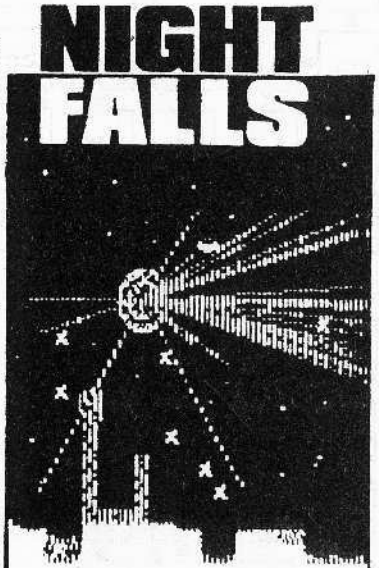
2) The grenade value appears as a letter or symbol. However, it still decrements by one for each grenade that you throw.

Neither of the above problems affects the play of the game, except to give you a lot of bullets and grenades.

ESCAPING CASTLE WOLFENSTEIN

The path out of Castle Wolfenstein is always the same. The contents of each room are randomized for each new game. Once this map is memorized it becomes easier to escape the Castle. Unfortunately, the plans are not guaranteed to be on the way out. My favorite tactic is to run for the exit, zapping or dodging as required and opening all chests I find along the way. If I haven't found the

a Stellar Vortex goes critical and Ends...



A single Stellar Vortex can utterly obliterate a cityscape. Fortunately they do not explode immediately upon appearance. They must grow to critical mass before they will explode. When about to explode they will begin to "tick" and change colors.

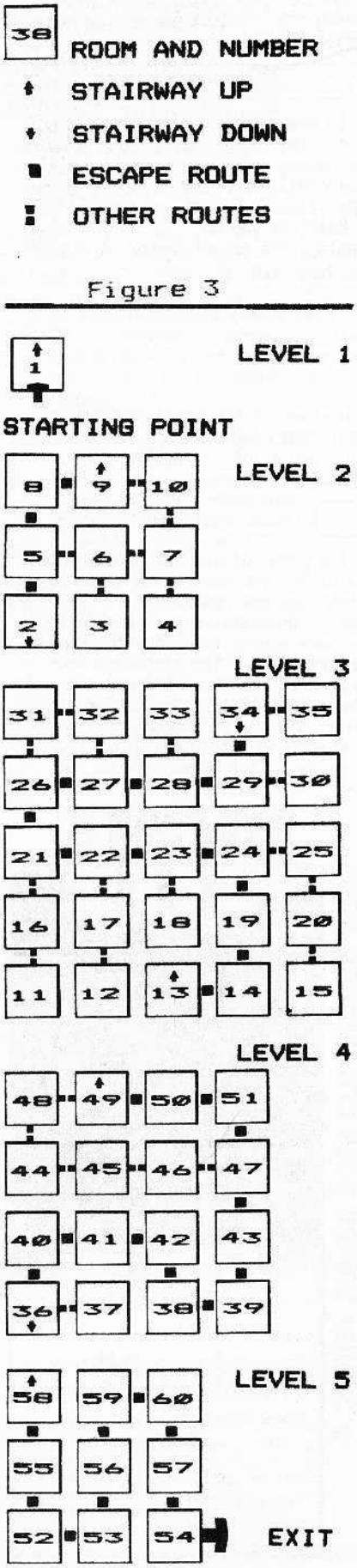


Figure 3

plans by the time I reach the last room, I backtrack and search until I find them.

GOOD LUCK AND HAVE FUN!!

Advanced Playing Techniques

The APT program is an alternate method to changing the variables in Castle Wolfenstein. Be sure to read "INSIDE CASTLE WOLFENSTEIN" before using this program.

Enter the program and save it to disk.

Using The Program

Insert your Castle Wolfenstein disk (unprotected) and run the program. You will first be asked for the file you wish to alter.

REMEMBER: change both BACKUP and CASTLE in the same way to insure that the variables are indeed changed (or ^BACKUP and ^CASTLE if these are on your disk instead).

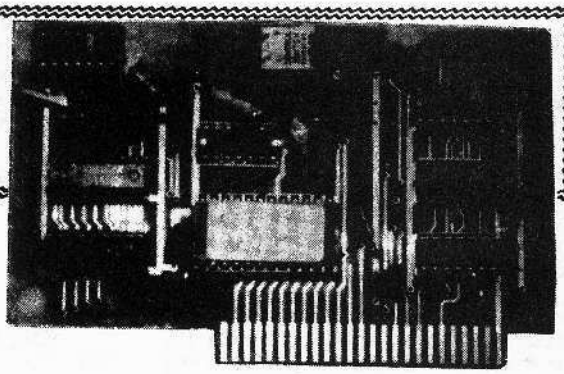
The default answer is Yes. ESC will return you to the main menu, or if pressed from the main menu will terminate the program. There is one additional choice provided, FIX BAD FILE. If when you are playing Castle Wolfenstein, the program freezes, try this option. It should restore the game and eliminate all SS who are actively pursuing you.

```
100  TEXT : HOME : NORMAL : HIMEM:
     16380
110  DATA 32,227,3,76,217,3
120  FOR X = 0 TO 5: READ Y: POKE
     768 + X,Y: NEXT
130  D$ =  CHR$ (13) +  CHR$ (4)
140  INVERSE : VTAB 2: HTAB 5: PRINT
     "A.P.T. FOR CASTLE WOLFENSTEIN":
     NORMAL
150  VTAB 7: HTAB 5: PRINT
     "1) CHANGE CASTLE": HTAB 5:
     PRINT "2) CHANGE BACKUP"
160  PRINT : HTAB 5: PRINT
     "3) CHANGE ^CASTLE": HTAB 5:
     PRINT "4) CHANGE ^BACKUP"
170  POKE 216,0: VTAB 13: HTAB 10:
     PRINT "WHICH ONE (1-4) "
     CHR$(7);: GET A$
180  IF A$ =  CHR$ (27) THEN  HOME :
     PRINT "PROGRAM TERMINATED": END
190  IF A$ < "1" OR A$ > "4" THEN 140
200  FI$ = "CASTLE": IF A$ = "2" OR
     A$ = "4" THEN FI$ = "BACKUP"
210  IF A$ > "2" THEN FI$ = "^" + FI$
220  ONERR  GOTO 170
230  PRINT D$"VERIFY"FI$: POKE 216,0:
     GOSUB 810
240  B1 = PEEK (TB)+ PEEK (TB + 1) *
     256
250  POKE TR, PEEK (B1 + 12): POKE
     SE, PEEK (B1 + 13)
260  POKE CMND,1: POKE BU,0: POKE BU
     + 1,64: POKE VOL,0: CALL 768
270  IF  PEEK (ERR) > 15 THEN 740
280  FI$ = "MAXIMUM GRENADES": GOSUB
     920 : IF B THEN POKE DB + 72,255
290  FI$ = "MAXIMUM BULLETS": GOSUB
     920: IF B THEN  POKE DB + 71,255
300  FI$ = "A UNIFORM": GOSUB 920: IF
     B THEN  POKE DB + 73,1
310  FI$ = "A BULLET PROOF VEST":
     GOSUB 920: IF B THEN  POKE DB +
     74,1
320  FI$ = "THE WAR PLANS": GOSUB 920
     : IF B THEN  POKE DB + 108,1
330  IF  PEEK (DB + 111) = 0 THEN 350
340  FI$ = "TO BE RESURECTED": GOSUB
     920 : IF B THEN  POKE DB + 111,0
350  HOME
360  FI$ = "TO CHANGE YOUR RANK":
     GOSUB 920: IF  NOT B THEN 450
370  PRINT : PRINT"CURRENT RANK IS ";
     INT (( PEEK (DB + 109) / 16 +
     1) / 2)
380  PRINT : POKE WL,5: PRINT : PRINT
     "1) PRIVATE": PRINT
     "2) CORPORAL"
390  PRINT "3) SERGEANT": PRINT
     "4) LIEUTENANT": PRINT
     "5) CAPTAIN"
400  PRINT "6) COLONEL": PRINT
     "7) GENERAL": PRINT
     "8) FIELD MARSHAL"
410  PRINT : PRINT "WHICH ONE (1-8) "
     ;: GET A$
420  POKE WL,0:A =  VAL (A$)
430  IF A < 1 OR A > 8 THEN  PRINT :
     PRINT : PRINT
     "MAINTAINING OLD RANK": FOR X =
     1 TO 500: NEXT : GOTO 450
440  POKE DB + 109,(2 * A - 1) * 16
450  HOME
460  FI$ = "TO CHANGE ROOMS": GOSUB
     920: IF  NOT B THEN 510
470  PRINT: PRINT"CURRENTLY IN ROOM "
     ; PEEK (DB + 64)
480  PRINT : INPUT
     "ENTER ROOM NUMBER (1-60) ";A$
490  B =  VAL (A$): IF B < 1 OR B >
     60 THEN  PRINT : PRINT
     "MAINTAINING OLD ROOM": FOR X =
     1 TO 500: NEXT : GOTO 510
500  POKE DB + 64,B
510  HOME
520  FI$ = "TO CHANGE PERCENT CHANCE
     OF ACHIEVING A HIT": GOSUB 920
530  IF  NOT B THEN 600
540  PRINT: PRINT"CURRENT CHANCE IS "
     ;: PRINT INT ( PEEK (DB + 75) /
     255 * 100);"%"
560  PRINT : INPUT
     "ENTER PERCENT WANTED ";A$
570  IF A$ = "" THEN 600
580  B =  VAL (A$): IF B < 0 OR B >
     100 THEN 600
590  POKE DB + 75,255 * B / 100
600  HOME
610  FI$ = "TO CHANGE PERCENT CHANCE
     OF BEING RECOGNIZED": GOSUB 920:
620  IF  NOT B THEN 680
630  PRINT:PRINT"CURRENT CHANCE IS ";
     INT ( PEEK (DB + 77) / 255)
640  PRINT : INPUT
     "CHANGE PERCENT CHANCE TO ";A$
650  IF A$ = "" THEN 680
660  B =  VAL (A$): IF B < 0 OR B >
     100 THEN 680
670  POKE DB + 77,255 * B / 100
680  HOME
```

Continued On Page 14

HARDWARE REVIEW:
the REPLAY card

Texas Ranch and Shoreline Systems



By Julie Joringdal

I am a beginning computerist. Because of this irrevocable and often frustrating fact, I was asked by the publisher of HardCore to write a review of the Replay card, from Texas Ranch and Shoreline Systems (TRASH), geared to the interests of other beginners who have found the going a little rough.

What You Get

Replay is a package which includes the Replay card itself, a utility disk, and a 57-page manual. The card enables the user to capture whatever program is in memory at the time the copy switch is flipped. The copy is written to disk automatically in a non-Apple compatible format. The utilities on the accompanying disk allow the copied program to be analyzed and edited, and packed into a normal DOS binary file.

Single Load Program

In order for a program to be copyable by the Replay card, it must be total load. This is a program which loads into memory when the disk is booted and doesn't need to access the disk again. Since most games are total load, Replay has some special advantages for game players. If you would like to avoid the lower skill levels, you can play a game up to whatever level you like and make a copy starting at that point. When the copy disk is booted, game play will start at the higher level.

Using The Card

The Replay card will copy absolutely any program executing in memory in 15 seconds. The card can be inserted in any slot, except slot 6 where the disk I/O card must be. The other requirement is that slot 0 either be empty or have the Replay card in it. (no RAM or ROM card can be present.)

Copies are made by first booting the program to be duplicated. When loading is completed, the disk is removed and a blank disk is inserted in the drive. The program in memory is frozen by flipping a switch on the Replay card (the top must be left off of the computer in order to reach the switch). A prompt warns that a copy is about to be made and the user is instructed to press any key to start the copying process. Remember to put a blank disk in the drive before you press a key. When a copy is completed, the words "Copy Made" appear on the monitor.

Running The Copy

Replay copies a program in a non-standard format which uses 10 sectors per track. A Replay copy will not boot with normal DOS. The Replay card must be in the computer. The copy is run by calling the card from BASIC or from the monitor. Two questions have to be answered before the program will run:

1. Whether to execute the program or just load it
2. Which text, LoRes, or HiRes page to display (either page 1 or 2)

The instructions will help to answer these questions. If the wrong page is selected, simply reboot.

Changing the copy

The editor utility allows the copy disk to be analyzed and edited. The edit buffer will hold up to 13 tracks at one time.

The contents of the buffer can be disassembled, altered, and written back to the disk.

Converting to Normal DOS

The packer is used in converting the Replay copy from its nonstandard format into a DOS 3.3 compatible file so it can be saved on a standard DOS disk. Having the copy on a standard DOS disk is an advantage because it allows more than one file to be saved on a disk.

The first step is to pack a binary file. The process involves keeping the parts of the program which are vital and throwing out the ones that aren't. This is necessary because the longest file that can be written to a DOS 3.3 disk is $80 pages. A Replay disk is $C0 pages, so it can't be converted as one long file.

An option is included which will pack a binary file automatically with the aid of a "command" file that contains the appropriate information.

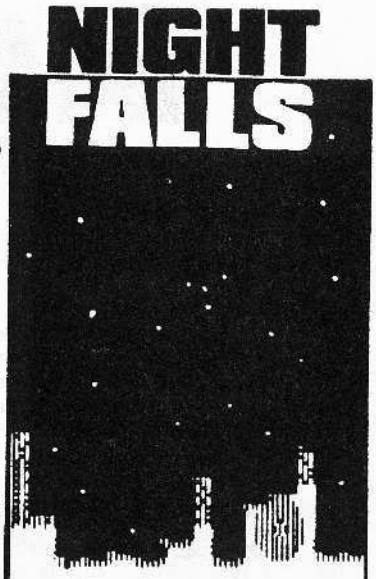
Several command files are included on the utility disk. Each file is specific for a certain program.

Documentation

Like most first-time documentation, the Replay manual contains many typographical errors which can often be confusing, especially if the user is a beginner and can't second-guess the author. Generally speaking, the editor and packer sections of the manual make assumptions aimed at the experienced user, while the Replay card section is easy enough for the beginner. I found the utility disk menu especially clear and helpful. It contains a brief explanation of each feature, which can serve to augment the manual and remind the user of program functions.

How it Rates

The card itself is easy to use. The



Demolished buildings & molten craters greet...

When the bombs and rays strike naked earth, they create huge craters which can destroy your cannons. Craters can be filled by successively sacrificing cannons until the craters are filled or by rebuilding over them.

importance of turning the computer off before inserting the card should be emphasized more clearly in the documentation. It would also have helped for the instuctions to point out when to remove the original disk and insert a blank one. Aside from this, the documentation concerning the use of the card is fairly easy to understand.

The directions given for the editor assume that the user knows what an editor is for and what to do with it. For example, the user is not told to remove the utility disk with the editor on it and insert the disk that the Replay copy is on. Most people will figure out that this needs to be done, but it would have been more helpful if this had been included in the documentation, along with when it should be done.

The packer utility was particularly difficult from the point of view of a beginner. A machine language background is needed to be able to recognize sections of code which are valid and other sections which can be left out. Examples of different kinds of code are provided, but to a beginner it is still a mystery.

The command files are handy for beginning and experienced users. In order to add your own command files, some machine language experience is again required. You must first pack the program successfully to have the correct information to be saved in the command file.

Conclusion

The card meets all of the claims of the manufacturer. It can be very useful to both the businessman and

CASTLE WOLFENSTEIN

Continued From Page 12

```
690  FI$ = "TO FIX A BAD FILE": GOSUB
     920: IF  NOT B THEN 710
700  FOR X = 110 TO 256: POKE DB + X
     ,0: NEXT : FOR X = 76 TO 106:
     POKE DB + X,0: NEXT
710  HOME : VTAB 12: HTAB 16: PRINT
     "WRITTING"
720  POKE CMND ,2: POKE VOL,0: CALL
     768: HOME
730  IF  PEEK (ERR) < 16 THEN 140
740  PRINT  CHR$ (7)
     "WARNING, DOS ERROR"
750  E =  PEEK (ERR)
760  IF E = 16 THEN  PRINT
     "WRITE PROTECTED (REMOVE TAB)"
770  IF E = 64 THEN PRINT
     "DRIVE ERROR (I/O)"
780  IF E < > 10 AND E < > 40 THEN
     PRINT "UNUSUAL ERROR, CODE = ";E
790  TEXT
800  END
810  FT = 46582:SL = 46583:DR = 46584
820  TR = 47084:SE = 47085
830  NS = 46574:TB = 46537
840  WL = 32:WW = 33:WT = 34
850  CMND = 47092:ERR = 47093
860  VOL = 47083
870  BU = 47088
880  DB = 16384
890  HOME : INVERSE : PRINT
     "FILE NAME:";
900  NORMAL : PRINT " ";FI$
910  POKE WT,5: VTAB 6: POKE ERR,0:
     RETURN
920  B = 0: PRINT "DO YOU WANT ";:
     POKE WL,12: INVERSE : PRINT
     FI$;: NORMAL : PRINT " (Y/N) ";:
     GET A$
930  POKE WL,0
940  IF A$ =  CHR$ (27) THEN  POP :
     GOTO 140
950  IF A$ =  CHR$ (13) THEN  PRINT :
     RETURN
960  PRINT A$
970  IF A$ = "Y" THEN B = 1
980  RETURN
```

the game player. For the businessman, it will provide a reliable backup and a quick method of loading a large program (for example, a calculation-type program and a favorite template could be saved together). In addition, the game player will find that the Replay card can open up a whole new range of player tactics.

IOB DATA STATEMENTS

Continued From Page 4

Line 60 gets the track data and line 70 POKEs it into the RWTS.

Line 110 sets the maximum number of sectors to copy for each track and changes a byte in the RWTS so that it will ignore checksums and end marks.

Line 120 will INITialize the target disk.

Line 1000 is changed to ignore the DOS tracks. (0-2)

Line 1010 changes the Input/Output CALL location and READs in a track.

Line 1015 looks for the VTOC track and sets the directory link to point to track 14, sector 4. This is because the protected disk has moved the true directory from it's normal location at track 17, sector C.

Line 1020 sets the I/O CALL back to normal DOS and writes the track info.

Line 63000 has been changed to POKE the additional bytes in line 63010 into page 3 of memory. This is the machine language subroutine which calls the appropriate RWTS.

Line 63035 BLOADs the 13 sector RWTS.

Lines 63050 and on are the decimal bytes that must be poked into the 13 sector RWTS in order to READ the protected disk.

Use FID to transfer the files from the unprotected version of Ultima to another disk so that the VTOC and directory will be on the correct track.

The hello name on the Ultima disk is "ULTIMA".

BOOT CODE TRACING

Continued From Page 8

Add the following relocation routine, so that when the program is BRUN everything will be put back in the proper place:

```
3ED0:00 00 A9 5D 85 3D A9 7F
3ED8:85 3F A9 9D 85 43 20 F3
3EE0:3E A9 3F 85 3D A9 4F 85
3EE8:3F A9 00 85 43 20 F3 3E
3EF0:4C 94 12 A0 FF 84 3E C8
3EF8:84 3C 84 42 20 2C FE 60
```

The source code for this routine looks like:

```
3ED2-A9 5D     LDA #$5D    Set up
3ED4-85 3D     STA $3D     address
3ED6-A9 7F     LDA #$7F    data to
3ED8-85 3F     STA $3F     move top
3EDA-A9 9D     LDA #$9D    part of
3EDC-85 43     STA $43     program.
3EDE-20 F3 3E  JSR $3EF3   Call move.
3EE1-A9 3F     LDA #$3F    Do it again
3EE3-85 3D     STA $3D     for the
3EE5-A9 4F     LDA #$4F    bottom
3EE7-85 3F     STA $3F     part of the
3EE9-A9 00     LDA #$00    program.
3EEB-85 43     STA $43
3EED-20 F3 3E  JSR $3EF3   Call move
3EF0-4C 94 12  JMP $1294   Run Program
3EF3-A0 FF     LDY #$FF    Get things
3EF5-84 3E     STY $3E     ready
3EF7-C8        INY         before
3EF8-84 3C     STY $3C     calling the
3EFA-84 42     STY $42     monitor
3EFC-20 2C FE  JSR $FE2C   move
3EFF-60        RTS         routine.
```

THE FINAL TEST

Finally, remove the protected disk from the drive and replace it with a normal DOS (slave) disk. Warm Boot it by typing:

```
6 CTRL-P
```

Do not type "CTRL-P", just hold the CTRL key down and press "P" then release the CTRL key and press "RETURN".

When the Boot is complete, type:

```
BSAVE MDC,A$3ED2,L$412E
```

You now have an unlocked program that will BRUN normally, or it can be customized as you see fit.

Try this procedure with your other "one-shot" load programs. You will probably be surprised at how often it works, with a little sleuthing. That is where the perseverance part comes in.

Page zero locations changed by RESET:

```
$20-$2B
$31
$33-$3F
$40-$49
$4E-$4F
```

SoftKey Publishing P.O. Box 44549 Tacoma, WA 98444

BACK ISSUES ARE AVAILABLE

Issues #1 and 2 (and soon, #3) are back-issues and, while they last, can be purchased at the sample copy rate:

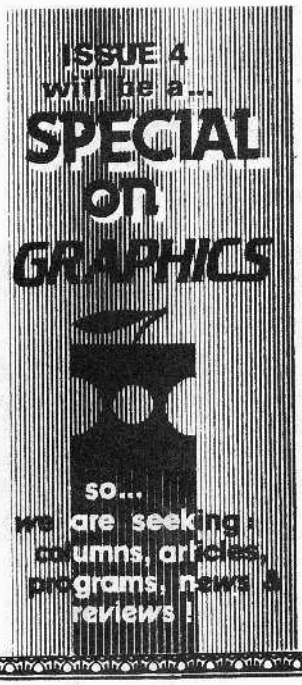
$5 each U.S. only
$8 each foreign orders

UPDATES, TOO

Also, back again because of popular demand (by those who want a complete collection of HARDCORE Computing) back issues of the Updates are also available:

$3 each U.S. only
$5 each foreign orders

U.S. Funds only
No credit cards
No COD's
No Phone Orders

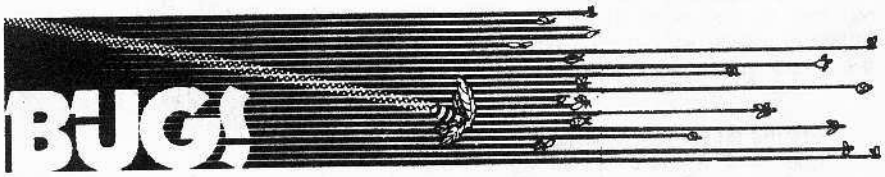


SoftKey

BULK RATE
U.S. Postage
PAID
Tacoma, WA
Permit No. 269

KIM KRUGLICK
265 MILLER AVE.
MILL VALLEY, CA 94941

HOT TIP: A reader called and left this message: "Change byte $B68C in DOS to 4C and speed up file SAVE times by 12%." This change will stop DOS from doing an automatic verify on every sector it writes. So, decreased SAVE times for a decreased reliability, the choice is up to you.



Due to last minute changes in the contents of Issue #3, the following corrections should be noted:

Page 1 - Table of Contents - VISICALC: "Job Costing" appears on page 51 instead of page 41. "HGR: Huey's Hi-Res Corner" appears on page 43 instead of page 42.

Page 33 - "Menu" - The machine language listing in the first column is correct, but to make line 0 identical to the sample line 0 change the two FF'S in line 828 to B2 84 by typing:
82C:B2 84
When setting the "End of Program" counter, make sure you type a space between the 3A and 08.

Page 40 - "Zephyr Wars" - Insert line 200:

```
200 ROT= 0:X% = XUF0%(Z) * 7:Y% =
    YUF0%(Z): IF Y% = 0 THEN RETURN
```

Page 46 - "Vector Graphics" - Line 210 should read:

```
210  IF A<0 THEN 250
```

Page 72 - "Bugs" - This section should read:
For those of you who have the magazine version of TEXT INVADERS, make this change to line 970 and it will stop crashing after you wipe out the first row of invaders.

```
970 INVADERS$(1) = LEFT$ (INVADERS$
    (1), RAN%) + TEMP$$ + MID$
    (INVADERS$ (AA), RAN% +2)
```

Also, for those with DiskView 1.1, there is a bug in line 690 of the "print routine". Please change line 690 to read:

```
690 PT = KY%: RETURN
```

And finally we have a correction in the disk version (as opposed to the HardCore issue #2 version) of Text Invaders. There is a colon (:) missing between the GOSUB 1530 and the NEXT A in line 60.