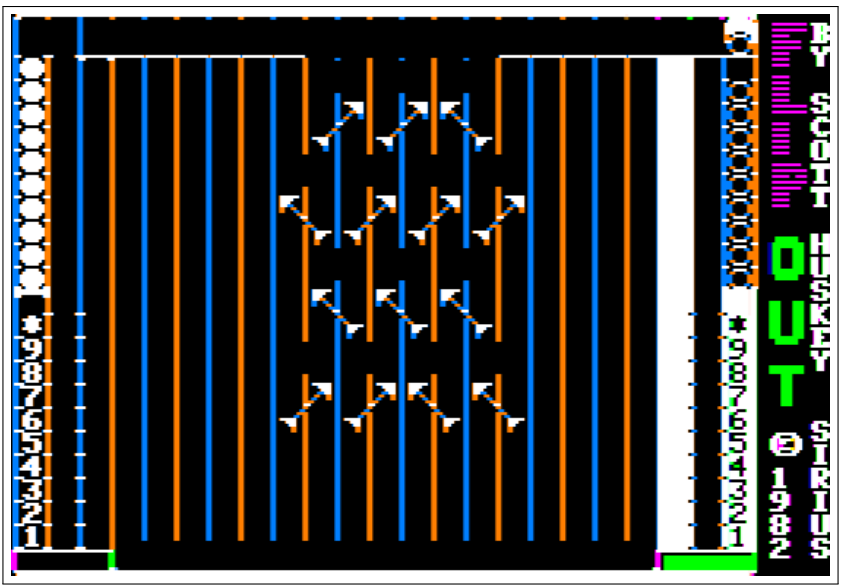
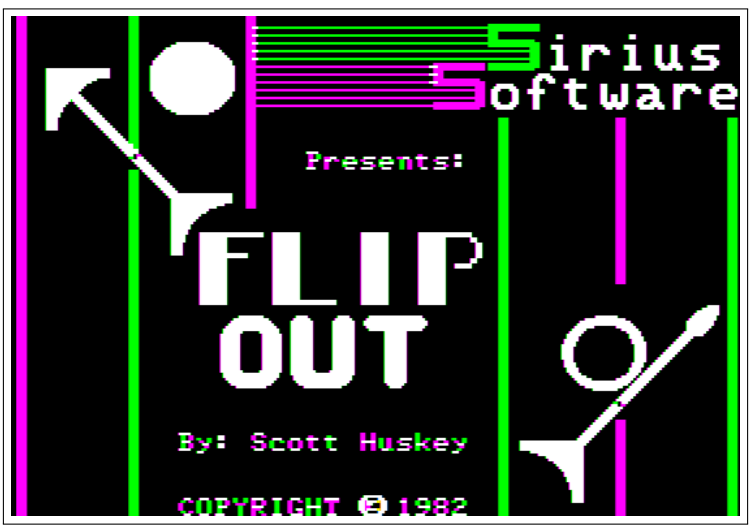
Flip Out



2015-08-04

Contents



----------Flip Out----------
A 4am crack 2015-08-04

Name: Flip Out
Genre: strategy
Year: 1982
Authors: Scott Huskey
Publisher: Sirius Software
Media: single-sided 5.25-inch floppy
OS: custom
Previous cracks: Captain Computer

> "One mustn't look at the abyss, because there is at the bottom an inexpressible charm which attracts us."
>
> Gustave Flaubert

Chapter 0
In Which Various Automated Tools Fail In Interesting Ways

COPYA
  immediate disk read error

Locksmith Fast Disk Backup
  unable to read any track

EDD 4 bit copy (no sync, no count)
  read errors on T0C-T1F, T21
  copy boots, loads several tracks,
  displays graphical title screen, then
  clears screen, displays "BOOT ERROR",
  and reboots

Copy][+ nibble editor
  can't make hide or hair of anything

Disk Fixer
  can't read anything beyond T00,S00
  under any combination of parameters

Why didn't COPYA work?
  not a 16-sector disk

Why didn't Locksmith FDB work?
  not a 16-sector disk

Why didn't my EDD copy work?
  I don't know. Probably a nibble check
  during boot.

The original disk displays the hi-res
title screen while loading. It is a
single-load game; it does not touch the
disk once it's fully loaded.

Next steps:

1. Trace the boot
2. Capture entire game in memory
3. Build a new disk with a fastloader to replicate the original disk's boot experience

Chapter 1
In Which We Start Off Loudly And Build To A Crescendo

```
[S6,D1=original disk]
[S5,D1=my work disk]

]PR#5
CAPTURING BOOT0
...reboots slot 6...
...reboots slot 5...
SAVING BOOT0

]BLOAD BOOT0,A$800
]CALL -151
*801L

; save boot slot number
0801-   A5 2B       LDA   $2B
0803-   AA          TAX
0804-   85 FB       STA   $FB
0806-   4A          LSR
0807-   4A          LSR
0808-   4A          LSR
0809-   4A          LSR
080A-   09 C0       ORA   #$C0
080C-   8D 00 30    STA   $3000

; zap language card
080F-   A0 00       LDY   #$00
0811-   84 00       STY   $00
0813-   A9 D0       LDA   #$D0
0815-   85 01       STA   $01
0817-   A2 30       LDX   #$30
0819-   AD 81 C0    LDA   $C081
081C-   AD 81 C0    LDA   $C081
081F-   B1 00       LDA   ($00),Y
0821-   91 00       STA   ($00),Y
0823-   C8          INY
0824-   D0 F9       BNE   $081F
0826-   E6 01       INC   $01
0828-   CA          DEX
0829-   D0 F4       BNE   $081F
```

```
; initialize globals
082B-   A6 FB       LDX   $FB
082D-   84 F7       STY   $F7
082F-   A9 04       LDA   #$04
0831-   85 F8       STA   $F8
0833-   85 FA       STA   $FA

; load some more sectors from track $00
; with a 4-4 encoding scheme and a
; prologue of "AD DA DD"
0835-   BD 8C C0    LDA   $C08C,X
0838-   10 FB       BPL   $0835
083A-   C9 AD       CMP   #$AD
083C-   D0 F7       BNE   $0835
083E-   BD 8C C0    LDA   $C08C,X
0841-   10 FB       BPL   $083E
0843-   C9 DA       CMP   #$DA
0845-   D0 F3       BNE   $083A
0847-   BD 8C C0    LDA   $C08C,X
084A-   10 FB       BPL   $0847
084C-   C9 DD       CMP   #$DD
084E-   D0 EA       BNE   $083A
0850-   A0 00       LDY   #$00
0852-   84 F5       STY   $F5
0854-   BD 8C C0    LDA   $C08C,X
0857-   10 FB       BPL   $0854
0859-   38          SEC
085A-   2A          ROL
085B-   85 F6       STA   $F6
085D-   B0 11       BCS   $0870

; main loop to read 2 nibbles and save
; 1 byte
085F-   BD 8C C0    LDA   $C08C,X
0862-   10 FB       BPL   $085F
0864-   2A          ROL
0865-   85 F6       STA   $F6
0867-   C8          INY
0868-   D0 06       BNE   $0870
```

```
; increment page
086A-   E6 F8        INC   $F8

; decrement sector count
086C-   C6 FA        DEC   $FA
086E-   F0 0F        BEQ   $087F
0870-   BD 8C C0     LDA   $C08C,X
0873-   10 FB        BPL   $0870
0875-   25 F6        AND   $F6
0877-   91 F7        STA   ($F7),Y

; calculate a running checksum
0879-   45 F5        EOR   $F5
087B-   85 F5        STA   $F5

; loop back for more bytes
087D-   B0 E0        BCS   $085F

; verify checksum
087F-   BD 8C C0     LDA   $C08C,X
0882-   10 FB        BPL   $087F
0884-   25 F6        AND   $F6
0886-   45 F5        EOR   $F5
0888-   D0 A5        BNE   $082F

; jump to the code we just loaded
088A-   4C 29 04     JMP   $0429
```

$F8 (initially 4) appears to be the page in memory to put the sector. It's incremented after each read (at $086A).

$FA (also initially 4) appears to be the sector count. It's decremented after each read (at $086C).

At $088A, it jumps to $0429 to continue the boot. So I need to capture the text page.

```
*9600<C600.C6FFM

; set up callback to my code after RWTS
; is loaded into text page
96F8-   A9 05       LDA   #$05
96FA-   8D 8B 08    STA   $088B
96FD-   A9 97       LDA   #$97
96FF-   8D 8C 08    STA   $088C

; start the boot
9702-   4C 01 08    JMP   $0801

; relocate RWTS to graphics page so it
; will survive a reboot
9705-   A2 04       LDX   #$04
9707-   A0 00       LDY   #$00
9709-   B9 00 04    LDA   $0400,Y
970C-   99 00 24    STA   $2400,Y
970F-   C8          INY
9710-   D0 F7       BNE   $9709
9712-   EE 0B 97    INC   $970B
9715-   EE 0E 97    INC   $970E
9718-   CA          DEX
9719-   D0 EE       BNE   $9709

; turn off slot 6 drive motor
971B-   AD E8 C0    LDA   $C0E8

; reboot to my work disk
971E-   4C 00 C5    JMP   $C500
```

```
*BSAVE TRACE1,A$9600,L$121
*9600G
...reboots slot 6...
...reboots slot 5...
]BSAVE BOOT1 0400-07FF,A$2400,L$400
```

Chapter 2
Just Because Your Paranoid Doesn't Mean They're Not Trying To Hack You

```
I'll need to leave this code at $2400
to list it. Relative branches will look
correct, but absolute jumps will be off
by $2000.

]CALL -151

*2429L

; TEXT mode
2429-   20 39 FB    JSR   $FB39

; reset input and output vectors
242C-   A9 F0       LDA   #$F0
242E-   85 36       STA   $36
2430-   A9 FD       LDA   #$FD
2432-   85 37       STA   $37
2434-   85 39       STA   $39
2436-   A9 1B       LDA   #$1B
2438-   85 38       STA   $38
243A-   8D 00 E8    STA   $E800

; memory copy
243D-   A0 00       LDY   #$00
243F-   B9 6D 07    LDA   $076D,Y
2442-   99 00 8F    STA   $8F00,Y
2445-   C8          INY
2446-   10 F7       BPL   $243F

; initialization
2448-   A0 00       LDY   #$00
244A-   84 FD       STY   $FD
244C-   84 F1       STY   $F1
244E-   84 F2       STY   $F2

; $3000 is the slot number (x16)
; (saved in boot0)
2450-   AD 00 30    LDA   $3000
2453-   8D 4E 8F    STA   $8F4E
```

```
; set vectors (BRK, reset, NMI, IRQ)
2456-   A9 07        LDA   #$07
2458-   8D F0 03     STA   $03F0
245B-   8D F2 03     STA   $03F2
245E-   8D FC 03     STA   $03FC
2461-   8D FE 03     STA   $03FE
2464-   A9 8F        LDA   #$8F
2466-   8D F1 03     STA   $03F1
2469-   8D F3 03     STA   $03F3
246C-   8D FD 03     STA   $03FD
246F-   8D FF 03     STA   $03FF
2472-   49 A5        EOR   #$A5
2474-   8D F4 03     STA   $03F4
2477-   A9 4C        LDA   #$4C
2479-   8D FB 03     STA   $03FB
247C-   A9 FB        LDA   #$FB
247E-   8D FA FF     STA   $FFFA
2481-   8D FC FF     STA   $FFFC
2484-   8D FE FF     STA   $FFFE
2487-   A9 03        LDA   #$03
2489-   8D FB FF     STA   $FFFB
248C-   8D FD FF     STA   $FFFD
248F-   8D FF FF     STA   $FFFF
```

That's a lot of paranoia right there. Like, all the paranoia.

```
; Even more paranoia: check if the byte
; we wrote to the language card RAM
; ($E800, set at $043A) is still there
; after we switch back to ROM. If it
; is, that means that something (like a
; modified F8 PROM) is interfering with
; the ROM/RAM softswitches and we're
; better off leaving "ROM" enabled
; because it's more likely to actually
; have the modifications we just made
; to all the low-level vectors at
; $FFFA..$FFFF. Out-faking the fakers.
2492-   AD 80 C0    LDA   $C080
2495-   AD 00 E8    LDA   $E800
2498-   C9 1B       CMP   #$1B
249A-   F0 03       BEQ   $249F
249C-   8D 81 C0    STA   $C081

; clear text screen 2
249F-   A9 A0       LDA   #$A0
24A1-   99 00 08    STA   $0800,Y
24A4-   99 00 09    STA   $0900,Y
24A7-   99 00 0A    STA   $0A00,Y
24AA-   99 00 0B    STA   $0B00,Y
24AD-   C8          INY
24AE-   D0 F1       BNE   $24A1

; show text screen 2
24B0-   AD 51 C0    LDA   $C051
24B3-   AD 55 C0    LDA   $C055

24B6-   A9 02       LDA   #$02
24B8-   A0 05       LDY   #$05
24BA-   8D 03 04    STA   $0403
24BD-   8C 02 04    STY   $0402
```

```
; read a track (not shown, but it uses
; a custom 4-4 encoding that stores 12
; sectors worth of data per track)
24C0-   20 02 05    JSR   $0502
24C3-   AD 03 04    LDA   $0403

; carry is clear on success
24C6-   90 18       BCC   $24E0

; retry to read the track ($0402 is a
; global number of retries across the
; entire disk -- if it hits 0, the disk
; is considered bad and it jumps to The
; Badlands)
24C8-   AC 02 04    LDY   $0402
24CB-   88          DEY
24CC-   F0 0F       BEQ   $24DD
24CE-   8C 02 04    STY   $0402
24D1-   20 1F 04    JSR   $041F
24D4-   AC 02 04    LDY   $0402
24D7-   AD 03 04    LDA   $0403
24DA-   4C BA 04    JMP   $04BA
24DD-   4C 02 8F    JMP   $8F02

; success path continues here --
; increment the track (stored as a
; phase, so increment it by 2 to get to
; the next whole track)
24E0-   69 02       ADC   #$02

; have we read track $06 yet?
24E2-   C9 0C       CMP   #$0C

; if not, skip over this
24E4-   90 0C       BCC   $24F2
```

```
; enough has been read from the disk to
; show the graphical title screen, so
; switch to hi-res screen 1
24E6-   8D 50 C0    STA   $C050
24E9-   8D 52 C0    STA   $C052
24EC-   8D 54 C0    STA   $C054
24EF-   8D 57 C0    STA   $C057

; execution continues here regardless,
; check if we're done completely
24F2-   C9 18       CMP   #$18

; loop until we've read everything
24F4-   D0 C2       BNE   $24B8

; don't know what these are yet
24F6-   20 82 05    JSR   $0582
24F9-   20 3A 06    JSR   $063A

; turn off drive motor
24FC-   BD 88 C0    LDA   $C088,X

; start game
24FF-   4C 63 8F    JMP   $8F63
```

It seems like we've loaded the entire game by the time we JSR to the routines at $0582 and $063A. They could be important (decrypting the game in memory or setting up some vital zero page locations). Or they could be pure copy protection. Or both. Only one way to find out.

Chapter 3
In Which We Detect The Matrix From Inside The Matrix

```
*2582L

; move drive head to track $21
2582-   A9 42        LDA   #$42
2584-   A6 FB        LDX   $FB
2586-   20 B1 05     JSR   $05B1

; look for a four-nibble sequence in
; the form "AA * * *", then count
; nibbles until another "AA"
2589-   BD 8E C0     LDA   $C08E,X
258C-   BD 8C C0     LDA   $C08C,X
258F-   10 FB        BPL   $258C
2591-   C9 AA        CMP   #$AA
2593-   D0 F7        BNE   $258C
2595-   BD 8C C0     LDA   $C08C,X
2598-   10 FB        BPL   $2595
259A-   BD 8C C0     LDA   $C08C,X
259D-   10 FB        BPL   $259A
259F-   BD 8C C0     LDA   $C08C,X
25A2-   10 FB        BPL   $259F
25A4-   C9 AA        CMP   #$AA
25A6-   F0 08        BEQ   $25B0
25A8-   E6 F1        INC   $F1
25AA-   D0 F3        BNE   $259F
25AC-   E6 F2        INC   $F2
25AE-   D0 EF        BNE   $259F
25B0-   60           RTS

*263AL

; move drive head to track $22
263A-   A9 44        LDA   #$44
263C-   A6 FB        LDX   $FB
263E-   20 B1 05     JSR   $05B1
```

```
; initialize counters
2641-   A9 04        LDA   ##$04
2643-   85 12        STA   $12
2645-   A9 00        LDA   ##$00
2647-   85 11        STA   $11
2649-   A9 08        LDA   ##$08
264B-   85 FE        STA   $FE
264D-   A0 00        LDY   ##$00
264F-   84 10        STY   $10

; look for a long nibble sequence
; "AA D5 D5 FF D6 FF FD FD DD EA B5 F7"
2651-   BD 8C C0     LDA   $C08C,X
2654-   10 FB        BPL   $2651
2656-   C9 AA        CMP   ##$AA
2658-   D0 F7        BNE   $2651
265A-   BD 8C C0     LDA   $C08C,X
265D-   10 FB        BPL   $265A
265F-   C9 D5        CMP   ##$D5
2661-   D0 F3        BNE   $2656
2663-   BD 8C C0     LDA   $C08C,X
2666-   10 FB        BPL   $2663
2668-   C9 D5        CMP   ##$D5
266A-   D0 EA        BNE   $2656
266C-   BD 8C C0     LDA   $C08C,X
266F-   10 FB        BPL   $266C
2671-   C9 FF        CMP   ##$FF
2673-   D0 E1        BNE   $2656
2675-   BD 8C C0     LDA   $C08C,X
2678-   10 FB        BPL   $2675
267A-   C9 D6        CMP   ##$D6
267C-   D0 D8        BNE   $2656
267E-   BD 8C C0     LDA   $C08C,X
2681-   10 FB        BPL   $267E
2683-   C9 FF        CMP   ##$FF
2685-   D0 CF        BNE   $2656
2687-   BD 8C C0     LDA   $C08C,X
268A-   10 FB        BPL   $2687
268C-   C9 FD        CMP   ##$FD
268E-   D0 C6        BNE   $2656
[...]
```

```
2690-   BD 8C C0     LDA   $C08C,X
2693-   10 FB        BPL   $2690
2695-   C9 FD        CMP   #$FD
2697-   D0 BD        BNE   $2656
2699-   BD 8C C0     LDA   $C08C,X
269C-   10 FB        BPL   $2699
269E-   C9 DD        CMP   #$DD
26A0-   D0 B4        BNE   $2656
26A2-   BD 8C C0     LDA   $C08C,X
26A5-   10 FB        BPL   $26A2
26A7-   C9 EA        CMP   #$EA
26A9-   D0 AB        BNE   $2656
26AB-   BD 8C C0     LDA   $C08C,X
26AE-   10 FB        BPL   $26AB
26B0-   C9 B5        CMP   #$B5
26B2-   D0 A2        BNE   $2656
26B4-   BD 8C C0     LDA   $C08C,X
26B7-   10 FB        BPL   $26B4
26B9-   C9 F7        CMP   #$F7
26BB-   D0 99        BNE   $2656

; decode some 4-4 encoded data
26BD-   BD 8C C0     LDA   $C08C,X
26C0-   10 FB        BPL   $26BD
26C2-   38           SEC
26C3-   2A           ROL
26C4-   85 F6        STA   $F6
26C6-   BD 8C C0     LDA   $C08C,X
26C9-   10 FB        BPL   $26C6
26CB-   25 F6        AND   $F6
26CD-   85 F6        STA   $F6
26CF-   BD 8C C0     LDA   $C08C,X
26D2-   10 FB        BPL   $26CF
26D4-   38           SEC
26D5-   2A           ROL
26D6-   85 FA        STA   $FA
26D8-   BD 8C C0     LDA   $C08C,X
26DB-   10 FB        BPL   $26D8
26DD-   25 FA        AND   $FA
26DF-   85 FA        STA   $FA
```

```
; compute a rolling checksum on a long
; sequence of nibbles
26E1-   BD 8C C0    LDA   $C08C,X
26E4-   10 FB       BPL   $26E1
26E6-   BD 8C C0    LDA   $C08C,X
26E9-   10 FB       BPL   $26E6
26EB-   45 10       EOR   $10
26ED-   85 10       STA   $10
26EF-   C8          INY
26F0-   D0 F4       BNE   $26E6
26F2-   C6 FE       DEC   $FE
26F4-   D0 F0       BNE   $26E6

; calculate a second rolling checksum
; from the final value of the first
; rolling checksum
26F6-   A5 10       LDA   $10
26F8-   45 11       EOR   $11
26FA-   85 11       STA   $11

; loop back and do it again, a total of
; 4 times (zero page $12 set at $0643)
26FC-   C6 12       DEC   $12
26FE-   F0 03       BEQ   $2703
2700-   4C 49 06    JMP   $0649

; check secondary checksum
2703-   A5 11       LDA   $11

; needs to be non-zero
2705-   D0 03       BNE   $270A

; ...or we jump to The Badlands
2707-   4C 02 8F    JMP   $8F02
```

There's more to this routine, but that is the meat of it:

1. find a long nibble prologue (that only appears once on the track)
2. checksum the following nibbles
3. do steps 1 and 2 repeatedly and make sure the checksum changes

This is the key point: the data being read from track $22 is non-repeatable. It's different every time it's read. How is that possible?

The prologue ("AA D5 D5 FF D6 FF FD FD DD EA B5 F7") looks important, but it's not. What's important is what comes after it, what's being checksummed over and over: a long sequence of zero bits. Because that is what is actually on the original disk: nothing.

When we say a "zero bit," we really mean "the lack of a magnetic state change." If the Disk II doesn't see a state change in a certain period of time, it calls that a "0". If it does see a change, it calls that a "1". But the drive can only tolerate a lack of state changes for so long -- about as long as it takes for two bits to go by.

Fun fact(*): this is why you need to use nibbles as an intermediate on-disk format in the first place. No valid nibble contains more than two zero bits consecutively, when written from most-significant to least-significant bit.

(*) not guaranteed, actual fun may vary

So what happens when a drive doesn't see a state change after the equivalent of two consecutive zero bits? The drive thinks the disk is weak, and it starts increasing the amplification to try to compensate, looking for a valid signal. But there is no signal. There is no data. There is just a yawning abyss of nothingness. Eventually, the drive gets desperate and amplifies so much that it starts returning random bits based on ambient noise from the disk motor and the magnetism of the Earth.

Seriously.

Returning random bits doesn't sound very useful for a storage medium, but it's exactly what the developer wanted, and that's exactly what this code is checking for. It's finding and reading and checksumming the same sequence of bits from the disk, over and over, and checking that they differ.

Bit copiers will never duplicate the long sequence of zero bits, because that's not what they read. Whatever randomness they get when they read the original disk will essentially get "frozen" onto the copy. The checksum of those frozen bits will always be the same, no matter how many times you read them. So the BNE at $0705 will never branch, and it will fall through to $0707 and jump to The Badlands.

God, I hate physical objects.

Chapter 4
In Which We Separate The Wheat From The Chaff

At this point, I'm almost certain that the routines at $0582 and $063A are pure copy protection. My failed EDD bit copy loaded the entire game into memory before choking and rebooting. I can't easily patch the boot1 code (loaded into $0400..$07FF), because

(a) it's 4-4 encoded with a custom prologue and I don't have a disk editor that could easily modify it, and
(b) boot0 checksums boot1 to ensure no evil hackers tampered with it

However, astute readers may notice that boot0 does not checksum itself. And there's plenty of empty space at the end of boot0 to patch boot1... immediately after it verifies that boot1 hasn't been patched.

[S6,D1=non-working EDD bit copy]

[Disk Fixer]
  ["O" for INPUT/OUTPUT CONTROL]
  [set CHECKSUM ENABLED = NO]

T00,S00
---------- DISASSEMBLY MODE ----------
008A:A9 60   LDA #$60 /
008C:8D 82 05 STA $0582 < added
008F:8D 3A 06 STA $063A \
0092:4C 29 04 JMP $0429   moved

This lets boot0 load boot1, then it patches the two routines at $0582 and $063A to immediately return (RTS) before continuing. Essentially, the disk is tracing and cracking itself. The calling code at $04F6 never checks the return value, so that should work. Theoretically.

]PR#6
...works...

I love it when practice matches theory.

Call that "Flip Out (4am crack).nib". I'm not done yet, but I know for sure that I understand the boot well enough to modify it and understand the copy protection well enough to bypass it.

Now let's make it awesome.

Chapter 5
In Which We Capture All The Things

First, let's zap all of memory with an unusual byte ($FD). This will allow me to verify memory range loaded by the bootloader.

```
*800:FD N 801<800.BEFEM

*9600<C600.C6FFM

; set up callback after boot0 loads and
; verifies boot1
96F8-   A9 05       LDA   #$05
96FA-   8D 8B 08    STA   $088B
96FD-   A9 97       LDA   #$97
96FF-   8D 8C 08    STA   $088C

; start the boot
9702-   4C 01 08    JMP   $0801

; callback is here --
; break to the monitor after the entire
; game is in memory
9705-   A9 59       LDA   #$59
9707-   8D 00 05    STA   $0500
970A-   A9 FF       LDA   #$FF
970C-   8D 01 05    STA   $0501
970F-   4C 29 04    JMP   $0429

*BSAVE TRACE2,A$9600,L$112
*9600G
...reboots slot 6...
...read read read...
<beep>
```

Poking around, it appears the game occupies $0C00..$8FFF. I'll save it in chunks.

```
*2000<C00.1FFFM
```

```
*C500G
...
]BSAVE OBJ.0C00-1FFF,A$2000,L$1400
]BRUN TRACE2
...
<beep>
*C500G
...
]BSAVE OBJ.2000-5FFF,A$2000,L$4000
]BRUN TRACE2
...
<beep>
*2000<6000.8FFFM
*C500G
...
]BSAVE OBJ.6000-8FFF,A$2000,L$3000
```

And, just for good measure, let's make sure I got it all:

```
]CALL -151

*800:FD N 801<800.BEFEM
*BLOAD OBJ.0C00-1FFF,A$C00
*BLOAD OBJ.2000-5FFF,A$2000
*BLOAD OBJ.6000-8FFF,A$6000
*8F63G
...works...
```

Almost there.

Chapter 6
If You Wish To Play A Game, You Must First Create The Universe

To reproduce the original disk's boot experience as faithfully as possible, I decided against releasing this as a file crack. It's 2015. Let's write a bootloader.

```
[S6,D1=blank formatted disk]
[S5,D1=my work disk]

]PR#5
]CALL -151

; page count (decremented)
0300-   A9 90       LDA   #$90
0302-   85 FF       STA   $FF

; logical sector (incremented)
0304-   A9 00       LDA   #$00
0306-   85 FE       STA   $FE

; call RWTS to write sector
0308-   A9 03       LDA   #$03
030A-   A0 88       LDY   #$88
030C-   20 D9 03    JSR   $03D9

; increment logical sector, wrap around
; from $0F to $00 and increment track
030F-   E6 FE       INC   $FE
0311-   A4 FE       LDY   $FE
0313-   C0 10       CPY   #$10
0315-   D0 07       BNE   $031E
0317-   A0 00       LDY   #$00
0319-   84 FE       STY   $FE
031B-   EE 8C 03    INC   $038C

; convert logical to physical sector
031E-   B9 40 03    LDA   $0340,Y
0321-   8D 8D 03    STA   $038D
```

```
; increment page to write
0324-   EE 91 03     INC   $0391

; loop until done with all $90 pages
0327-   C6 FF        DEC   $FF
0329-   D0 DD        BNE   $0308
032B-   60           RTS

; logical to physical sector mapping
*340.34F

0340- 00 07 0E 06 0D 05 0C 04
0348- 0B 03 0A 02 09 01 08 0F

; RWTS parameter table, pre-initialized
; with slot 6, drive 1, track $01,
; sector $00, address $0C00, and RWTS
; write command ($02)
*388.397

0388- 01 60 01 00 01 00 FB F7
0390- 00 0C 00 00 02 00 00 60

*BSAVE MAKE,A$300,L$98

*800:0 N 801<800.BEFEM   ; clear memory
*BLOAD OBJ.0C00-1FFF,A$C00
*BLOAD OBJ.2000-5FFF,A$2000
*BLOAD OBJ.6000-8FFF,A$6000

*300G          ; write game to disk
```

Now I have the entire game on tracks $01-$09 of a standard format disk.

The bootloader (which I've named 4boot) lives on track $00. T00,S00 is boot0, which reuses the disk controller ROM routine to load boot1, which lives on sectors $0C-$0E.

Boot0 looks like this:

```
; decrement sector count
0801-   CE 19 08    DEC   $0819

; branch once we've read enough sectors
0804-   30 12       BMI   $0818

; increment physical sector to read
0806-   E6 3D       INC   $3D

; set page to save sector data
0808-   A9 BF       LDA   #$BF
080A-   85 27       STA   $27

; decrement page
080C-   CE 09 08    DEC   $0809

; $0880 is a sparse table of $C1..$C6,
; so this sets up the proper jump to
; the disk controller ROM based on the
; slot number
080F-   BD 80 08    LDA   $0880,X
0812-   8D 17 08    STA   $0817

; read a sector (exits via $0801)
0815-   4C 5C 00    JMP   $005C

; sector read loop exits to here (from
; $0804) -- note: by the time execution
; reaches here, $0819 is $FF, so this
; just resets the stack
0818-   A2 03       LDX   #$03
081A-   9A          TXS
```

```
; set up zero page (used by RWTS) and
; push an array of addresses to the
; stack at the same time
081B-   A2 0F       LDX   #$0F
081D-   BD 80 08    LDA   $0880,X
0820-   95 F0       STA   $F0,X
0822-   48          PHA
0823-   CA          DEX
0824-   D0 F7       BNE   $081D
0826-   60          RTS

*881.88F

0880-    88 FE 92 FE 2E FB FF
0888- BC 62 8F 0C 09 00 00 00
```

These are pushed to the stack in reverse order, starting with $088F. When we hit the "RTS" at $0826, it pops the stack and jumps to $FE89, then $FE93, then $FB2F, then $BD00, then $8F63.

- $FE89, $FE93, and $FB2F are in ROM (reset input, output, and textmode)
- $BD00 is the RWTS entry point. It loads T01-T09 into memory, starting at $0C00. (These values are stored in zero page, which we just set.)
- $8F63 is the game entry point. It never returns, so the other values on the stack are irrelevant.

The RWTS at $BD00 is derived from the ProDOS RWTS. It uses in-place nibble decoding to avoid extra memory copying, and it uses "scatter reads" to read whatever sector is under the drive head when it's ready to load something.

```
*BD00L

; set up some places later in the RWTS
; where we need to read from a slot-
; specific data latch
BD00-   A6 2B       LDX   $2B
BD02-   8A          TXA
BD03-   09 8C       ORA   #$8C
BD05-   8D 96 BD    STA   $BD96
BD08-   8D AD BD    STA   $BDAD
BD0B-   8D C3 BD    STA   $BDC3
BD0E-   8D D7 BD    STA   $BDD7
BD11-   8D EC BD    STA   $BDEC

; advance drive head to next track
BD14-   20 53 BE    JSR   $BE7C
```

$BE7C is actually a wrapper around the advance-drive-head routine. The real routine starts at $BE53. It looks like this:

```
*BE7CL

; advance drive head
BE7C-   20 53 BE    JSR   $BE53

; check current phase (track x2)
BE7F-   A5 FD       LDA   $FD
BE81-   C9 0A       CMP   #$0A
BE83-   D0 0C       BNE   $BE91

; once we've read enough into memory,
; show the graphical title screen
BE85-   2C 54 C0    BIT   $C054
BE88-   2C 57 C0    BIT   $C057
BE8B-   2C 52 C0    BIT   $C052
BE8E-   2C 50 C0    BIT   $C050
BE91-   60          RTS
```

```
This reproduces the behavior of the
original disk's loader, which showed
the title screen briefly while it
continued loading the rest of the game.

Continuing at $BD17...

; sectors-left-to-read-on-this-track
; counter
BD17-   A0 0F       LDY   #$0F
BD19-   84 F8       STY   $F8

; Initialize array at $0100 that tracks
; which sectors we've read from the
; current track. The array is in
; physical sector order, thus the RWTS
; assumes data is stored in physical
; sector order on each track. Values
; are the actual pages in memory where
; that sector should go, and they get
; zeroed once the sector is read.
BD1B-   98          TYA
BD1C-   18          CLC
BD1D-   65 FB       ADC   $FB
BD1F-   99 00 01    STA   $0100,Y
BD22-   88          DEY
BD23-   10 F6       BPL   $BD1B

; find the next address prologue and
; store the address field in $2C..$2F,
; like DOS 3.3
BD25-   20 0F BE    JSR   $BE0F

; check if this sector has been read
BD28-   A4 2D       LDY   $2D
BD2A-   B9 00 01    LDA   $0100,Y

; if 0, we've read this sector already,
; so loop back and look for another
BD2D-   F0 F6       BEQ   $BD25
```

```
; if not 0, use the target page and set
; up some STA instructions in the RWTS
; so we write this sector directly to
; its intended page in memory
BD2F-   A8          TAY
BD30-   84 FF       STY   $FF
BD32-   8C EA BD    STY   $BDEA
BD35-   A5 FE       LDA   $FE
BD37-   8D E9 BD    STA   $BDE9
BD3A-   38          SEC
BD3B-   E9 54       SBC   #$54
BD3D-   8D D1 BD    STA   $BDD1
BD40-   B0 02       BCS   $BD44
BD42-   88          DEY
BD43-   38          SEC
BD44-   8C D2 BD    STY   $BDD2
BD47-   E9 57       SBC   #$57
BD49-   8D AA BD    STA   $BDAA
BD4C-   B0 01       BCS   $BD4F
BD4E-   88          DEY
BD4F-   8C AB BD    STY   $BDAB

; read the sector into memory
BD52-   20 6D BD    JSR   $BD6D

; if that failed, just loop back and
; look for another sector
BD55-   B0 CE       BCS   $BD25

; mark this sector as read
BD57-   A4 2D       LDY   $2D
BD59-   A9 00       LDA   #$00
BD5B-   99 00 01    STA   $0100,Y
BD5E-   E6 FB       INC   $FB

; decrement sectors-left-to-read-on-
; this-track counter
BD60-   C6 F8       DEC   $F8
```

```
; loop until we've read all the sectors
; on this track
BD62-   10 C1        BPL   $BD25

; decrement tracks-left-to-read counter
; (set in boot0)
BD64-   C6 FC        DEC   $FC

; loop until we've read all the tracks
BD66-   D0 AC        BNE   $BD14

; turn off drive motor and exit
BD68-   BD 88 C0     LDA   $C088,X
BD6B-   38           SEC
BD6C-   60           RTS
```

Quod erat liberandum.

A 4am crack No. 392

---EOF---