Plasmania

2014-08-01

Contents

Plasmania

A 4am crack 2014-08-01

"Plasmania" is a 1983 arcade game by Lewis Geer and distributed by Sirius Software, Inc. It is notable for its speech synthesis routine on load, and at the start of each game, that says "Sirius presents Plasmania, ha ha ha ha ha" through the Apple speaker.

The original disk loads the entire game into memory in one shot, but when you select "Start Game" from the main menu, it accesses the disk again. It might be possible to capture it as a file. Or it might not. One never knows, do they?

COPYA fails miserably and immediately with a disk read error. EDD 4 bit copy produces a copy that appears to load (lots of disk activity, including loading a graphical title screen), but then the disk grinds several times and reboots with a "BOOT ERROR" message.

In my experience, programs do not spontaneously reboot unless someone tells them to.

My trusty Copy][+ sector editor can't make heads or tails of anything beyond T00,S00. In fact, even the raw nibbles look weird.

```
                    --v--

   COPY ][ PLUS BIT COPY PROGRAM 8.4
(C) 1982-9 CENTRAL POINT SOFTWARE, INC.
----------------------------------------

TRACK: 01  START: 1A00  LENGTH: 1634

2808: F7 F7 F7 F7 F7 F7 F7 F7   VIEW
2810: F7 F7 F7 F7 F7 F7 F7 F7
2818: F7 F7 F7 F7 F7 F7 F7 F7
2820: F7 F7 F7 F7 F7 F7 F7 F7
2828: F7 F7 F7 F7 F7 F7 F7 AD  <-282F
2830: DA DD FA FA FF BF EE AF
2838: FB FB FF BF AE EE AA AA
2840: BA EA AE AA AA AF AA AA
2848: FE AF FB EA BE AA FF FF

----------------------------------------

  A  TO ANALYZE DATA  ESC TO QUIT

  ?  FOR HELP SCREEN  /  CHANGE PARMS

  Q  FOR NEXT TRACK   SPACE TO RE-READ

                    --^--
```

That "<-282F" is pointing to where Copy][+ thinks the "start" of the track is, based on its "analyze data" feature. It doesn't look anything like a normal 16-sector address prologue. I'm guessing that this is 4-4 encoded data.

Time for boot tracing with AUTOTRACE.

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

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

For those of you just tuning in, my
work disk uses a custom program that I
affectionately call "AUTOTRACE" to
automate the process of boot tracing as
far as possible. For some disks (like
this one, apparently), it just captures
track 0, sector 0 (saved in a file
called "BOOT0") and stops. For other
disks that load in the same way that an
unprotected DOS 3.3 disk loads, it
captures the next stage of the boot
process as well (in a file called
"BOOT1"). But in this case, it stopped
after capturing T00,S00, so I need to
look at that code and figure out how
this disk boots.

]CALL -151

*800<2800.28FFM

*801L

; display hi-res graphics page
; (uninitialized)
0801-   AD 50 C0    LDA   $C050
0804-   AD 52 C0    LDA   $C052
0807-   AD 54 C0    LDA   $C054
080A-   AD 57 C0    LDA   $C057
```

```
; set reset and BRK vectors
080D-   A9 08       LDA   #$08
080F-   8D F3 03    STA   $03F3
0812-   8D F1 03    STA   $03F1
0815-   49 A5       EOR   #$A5
0817-   8D F4 03    STA   $03F4
081A-   A9 CF       LDA   #$CF
081C-   8D F2 03    STA   $03F2
081F-   8D F0 03    STA   $03F0

; determine where the disk controller
; ROM routine is, based on the slot
; number x16 (in X register at boot)
0822-   8A          TXA
0823-   85 05       STA   $05
0825-   4A          LSR
0826-   4A          LSR
0827-   4A          LSR
0828-   4A          LSR
0829-   09 C0       ORA   #$C0
082B-   8D FC 08    STA   $08FC

; wipe memory from $0900..$BFFF
082E-   A0 00       LDY   #$00
0830-   84 06       STY   $06
0832-   A9 09       LDA   #$09
0834-   85 07       STA   $07
0836-   85 02       STA   $02
0838-   A2 B7       LDX   #$B7
083A-   A9 A0       LDA   #$A0
083C-   91 06       STA   ($06),Y
```

```
; if wipe fails, even for a single
; byte, jump to The Badlands and reboot
083E-   D1 06        CMP   ($06),Y
0840-   F0 03        BEQ   $0845
0842-   4C CF 08     JMP   $08CF
0845-   C8           INY
0846-   D0 F2        BNE   $083A
0848-   E6 07        INC   $07
084A-   CA           DEX
084B-   D0 ED        BNE   $083A
084D-   8C 00 02     STY   $0200

; wipe language card
0850-   A9 D0        LDA   #$D0
0852-   85 07        STA   $07
0854-   A2 30        LDX   #$30
0856-   AD 81 C0     LDA   $C081
0859-   AD 81 C0     LDA   $C081
085C-   B1 06        LDA   ($06),Y
085E-   91 06        STA   ($06),Y
0860-   C8           INY
0861-   D0 F9        BNE   $085C
0863-   E6 07        INC   $07
0865-   CA           DEX
0866-   D0 F4        BNE   $085C

; The rest of boot0 appears to load the
; boot1 code from track 0 using a 4-4
; nibble encoding scheme. Zero page $05
; contains the slot number x16 (saved
; earlier, at $0823).
0868-   A6 05        LDX   $05

; starting address for data ($0400)
086A-   A9 04        LDA   #$04
086C-   85 07        STA   $07
```

```
; number of sectors to read (4)
086E-   A9 04        LDA   #$04
0870-   85 01        STA   $01

; read/decode the data
0872-   A0 00        LDY   #$00
0874-   BD 8C C0     LDA   $C08C,X
0877-   10 FB        BPL   $0874
0879-   C9 AD        CMP   #$AD
087B-   D0 F7        BNE   $0874
087D-   BD 8C C0     LDA   $C08C,X
0880-   10 FB        BPL   $087D
0882-   C9 DA        CMP   #$DA
0884-   D0 F3        BNE   $0879
0886-   BD 8C C0     LDA   $C08C,X
0889-   10 FB        BPL   $0886
088B-   C9 DD        CMP   #$DD
088D-   D0 EA        BNE   $0879
088F-   84 00        STY   $00
0891-   BD 8C C0     LDA   $C08C,X
0894-   10 FB        BPL   $0891
0896-   38           SEC
0897-   2A           ROL
0898-   85 04        STA   $04
089A-   B0 12        BCS   $08AE
089C-   BD 8C C0     LDA   $C08C,X
089F-   10 FB        BPL   $089C
08A1-   38           SEC
08A2-   2A           ROL
08A3-   85 04        STA   $04
08A5-   C8           INY
08A6-   D0 06        BNE   $08AE
08A8-   E6 07        INC   $07
08AA-   C6 01        DEC   $01
08AC-   F0 0F        BEQ   $08BD
08AE-   BD 8C C0     LDA   $C08C,X
08B1-   10 FB        BPL   $08AE
[...]
```

```
08B3-   25 04        AND   $04
08B5-   91 06        STA   ($06),Y
08B7-   45 00        EOR   $00
08B9-   85 00        STA   $00
08BB-   B0 DF        BCS   $089C
08BD-   BD 8C C0     LDA   $C08C,X
08C0-   10 FB        BPL   $08BD
08C2-   25 04        AND   $04
08C4-   45 00        EOR   $00
08C6-   D0 03        BNE   $08CB

; jump to boot1
08C8-   4C 00 04     JMP   $0400
08CB-   C6 02        DEC   $02
08CD-   D0 9B        BNE   $086A

; The Badlands -- from which there is
; no return (clears the screen and
; prints "BOOT ERROR" and reboots)
08CF-   20 58 FC     JSR   $FC58
08D2-   A0 00        LDY   #$00
08D4-   B9 DF 08     LDA   $08DF,Y
08D7-   F0 11        BEQ   $08EA
08D9-   99 00 04     STA   $0400,Y
08DC-   C8           INY
08DD-   D0 F5        BNE   $08D4
...
08FA-   4C 00 C6     JMP   $C600
```

OK, it looks like I need to capture the boot1 code that's loaded into the text page at $0400..$07FF. Of course, this boot0 isn't going to make that easy, since it wipes basically everything from memory before loading. It looks like neutering the STA at $083C and the JMP at $0842 will be enough to bypass that.

```
*9600<C600.C6FFM

; neutralize the memory wiping
96F8-   A9 24       LDA   #$24
96FA-   8D 3C 08    STA   $083C
96FD-   8D 5E 08    STA   $085E

; neutralize the check for whether the
; memory wiping succeeded
9700-   A9 2C       LDA   #$2C
9702-   8D 42 08    STA   $0842

; set up a callback after loading boot1
; so it jumps to code under my control
; instead of continuing to $0400
9705-   A9 12       LDA   #$12
9707-   8D C9 08    STA   $08C9
970A-   A9 97       LDA   #$97
970C-   8D CA 08    STA   $08CA

; start the boot
970F-   4C 01 08    JMP   $0801

; callback is here --
; copy the boot1 code from the text
; page to higher memory so it will
; survive a reboot
9712-   A2 04       LDX   #$04
9714-   A0 00       LDY   #$00
9716-   B9 00 04    LDA   $0400,Y
9719-   99 00 24    STA   $2400,Y
971C-   C8          INY
971D-   D0 F7       BNE   $9716
971F-   EE 18 97    INC   $9718
9722-   EE 1B 97    INC   $971B
9725-   CA          DEX
9726-   D0 EE       BNE   $9716
```

```
; turn off the slot 6 drive motor
9728-   AD E8 C0    LDA   $C0E8

; reboot to my work disk
972B-   4C 00 C5    JMP   $C500

*BSAVE TRACE1,A$9600,L$12E

*9600G
...reboots slot 6...
...reboots slot 5...

]BSAVE BOOT1 0400-07FF,A$2400,L$400
]CALL -151

(Remember, this code expects to be at
$0400, not $2400. Relative branches
will look correct, but absolute
addresses will be off by $2000.)

*2400L

; Set up a jump table at $BFF0 (for
; what?)
2400-   A9 85       LDA   #$85
2402-   8D F1 BF    STA   $BFF1
2405-   A9 05       LDA   #$05
2407-   8D F2 BF    STA   $BFF2
240A-   A9 CD       LDA   #$CD
240C-   8D F4 BF    STA   $BFF4
240F-   A9 06       LDA   #$06
2411-   8D F5 BF    STA   $BFF5
2414-   A9 4C       LDA   #$4C
2416-   8D F0 BF    STA   $BFF0
2419-   8D F3 BF    STA   $BFF3
```

```
; Set up the sector read routine with
; the slot number (from zero page $05).
241C-   A5 05       LDA   $05
241E-   8D E4 07    STA   $07E4
2421-   4A          LSR
2422-   4A          LSR
2423-   4A          LSR
2424-   4A          LSR
2425-   09 C0       ORA   #$C0
2427-   8D D7 07    STA   $07D7

; A little piece of "f--- you" code
; here to deter boot tracers, I guess.
; Boot0 stored a $00 in $200, the start
; of the input buffer (used by BASIC
; and by the monitor). So if you
; interrupted the boot before now, then
; tried to run this code manually from
; the monitor, this will detect the
; disturbance and jump to The Badlands.
242A-   AD 00 02    LDA   $0200
242D-   F0 03       BEQ   $2432
242F-   4C 8F 07    JMP   $078F
2432-   A9 00       LDA   #$00
2434-   8D EB 07    STA   $07EB
```

```
; Aggressively set all possible vectors
; to jump to The Badlands, including:
; - BRK vector ($03EF)
; - reset vector ($03F2)
; - ampersand vector ($03F5)
; - Ctrl-Y vector ($03F8)
; - maskable interrupt vector ($03FE)
; - non-maskable interrupt
;   vector ($03FB)
; - in-ROM reset vector ($FFFC)
; - in-ROM maskable interrupt
;   vector ($FFFE)
; - in-ROM non-maskable interrupt
;   vector ($FFFA)
2437-   A9 8F       LDA   #$8F
2439-   8D F0 03    STA   $03F0
243C-   8D F2 03    STA   $03F2
243F-   8D F6 03    STA   $03F6
2442-   8D F9 03    STA   $03F9
2445-   8D FC 03    STA   $03FC
2448-   8D FE 03    STA   $03FE
244B-   8D FA FF    STA   $FFFA
244E-   8D FC FF    STA   $FFFC
2451-   8D FE FF    STA   $FFFE
2454-   A9 07       LDA   #$07
2456-   8D F1 03    STA   $03F1
2459-   8D F3 03    STA   $03F3
245C-   8D F7 03    STA   $03F7
245F-   8D FA 03    STA   $03FA
2462-   8D FD 03    STA   $03FD
2465-   8D FF 03    STA   $03FF
2468-   8D FB FF    STA   $FFFB
246B-   8D FD FF    STA   $FFFD
246E-   8D FF FF    STA   $FFFF
2471-   49 A5       EOR   #$A5
2473-   8D F4 03    STA   $03F4

; switch to read-only for language card
2476-   AD 80 C0    LDA   $C080
```

```
; This is the main loop that reads the
; entire game into memory. It reads a
; track at a time, but because of the
; non-standard encoding, each track
; only holds 12 sectors instead of 16.
2479-   A9 05       LDA   #$05
247B-   8D F7 07    STA   $07F7
247E-   8D F8 07    STA   $07F8
2481-   8D F9 07    STA   $07F9

; current track (x2)
2484-   A9 02       LDA   #$02
2486-   8D ED 07    STA   $07ED
2489-   4A          LSR
248A-   A8          TAY

; get starting address for the data to
; read from this track (array is below)
248B-   B9 F7 04    LDA   $04F7,Y
248E-   85 FD       STA   $FD
2490-   A9 00       LDA   #$00
2492-   85 FC       STA   $FC
2494-   AD ED 07    LDA   $07ED
2497-   AE E4 07    LDX   $07E4

; seek to track
249A-   20 CE 06    JSR   $06CE

; wait loop
249D-   A9 64       LDA   #$64
249F-   20 D8 07    JSR   $07D8

; read track
24A2-   A9 0C       LDA   #$0C
24A4-   8D EE 07    STA   $07EE
24A7-   20 3A 06    JSR   $063A

; read succeeded, continue
24AA-   90 03       BCC   $24AF
```

```
; read failed, jump to some code that
; checks whether there have been too
; many read failures, and if so, jump
; to The Badlands (otherwise jumps
; back to $0484 to start over)
24AC-   4C 6C 07    JMP   $076C

; increment track
24AF-   EE ED 07    INC   $07ED
24B2-   EE ED 07    INC   $07ED

; done yet?
24B5-   AD ED 07    LDA   $07ED
24B8-   C9 22       CMP   #$22

; nope, read more
24BA-   90 CD       BCC   $2489

; seek to track $11
24BC-   A9 22       LDA   #$22
24BE-   8D ED 07    STA   $07ED
24C1-   20 CE 06    JSR   $06CE

; wait
24C4-   A9 FF       LDA   #$FF
24C6-   20 D8 07    JSR   $07D8

; This is some sort of nibble check on
; tracks $11-$14. Every track needs to
; pass the test, otherwise...
24C9-   20 08 05    JSR   $0508
24CC-   20 E5 05    JSR   $05E5
24CF-   90 03       BCC   $24D4

; ...jump to The Badlands
24D1-   4C 77 07    JMP   $0777
24D4-   EE ED 07    INC   $07ED
24D7-   EE ED 07    INC   $07ED
24DA-   AD ED 07    LDA   $07ED
24DD-   C9 26       CMP   #$26
```

```
; OK, enough of this, let's go
24DF-   B0 10       BCS   $24F1
24E1-   20 CE 06    JSR   $06CE
24E4-   A9 FF       LDA   #$FF
24E6-   20 D8 07    JSR   $07D8
24E9-   A9 0A       LDA   #$0A
24EB-   8D F8 07    STA   $07F8
24EE-   4C C9 04    JMP   $04C9

; success path is here --
; turn off the drive motor and jump to
; the start of the game
24F1-   BD 88 C0    LDA   $C088,X
24F4-   4C 00 60    JMP   $6000

; array of addresses for reads (loaded
; at $048B)
24F8-   08 14 20 2C 38 40 4C 54
2500-   60 6C 78 84 90 9C A8 B3
```

If I'm doing this math correctly, this game takes up essentially all of main memory, $0800..$BEFF. ($Cx00 is for I/O and slots firmware, and $D000 and up is either ROM or language card RAM.) I guess that explains why they had to put the RWTS in the text page!

A thought: even if this game uses all all that memory, it still might be possible to capture it as a single file. There are "64K" DOS modifications that store most of the DOS in the language card and just keep a small (256 byte) stub in main memory. I use this on my standard work disk, because it makes it easier to work with disks that load things into normally reserved ranges during boot.

Another thought: if the boot loader fills up the entire main memory, what the heck is the game loading from disk when you select "Start Game"? I'm guessing that's pure copy protection, so I'll need to find it and bypass it.

```
*96F8L

; same as previous trace -- neutralize
; memory wipes and other unpleasantries
; so I can set up a callback after the
; RWTS is in memory
96F8-   A9 24       LDA   #$24
96FA-   8D 3C 08    STA   $083C
96FD-   8D 5E 08    STA   $085E
9700-   A9 2C       LDA   #$2C
9702-   8D 42 08    STA   $0842
9705-   A9 12       LDA   #$12
9707-   8D C9 08    STA   $08C9
970A-   A9 97       LDA   #$97
970C-   8D CA 08    STA   $08CA

; start the boot
970F-   4C 01 08    JMP   $0801

; at the success path ($04F1), switch
; back to ROM and jump to the monitor
9712-   A9 AD       LDA   #$AD
9714-   8D F1 04    STA   $04F1
9717-   A9 82       LDA   #$82
9719-   8D F2 04    STA   $04F2
971C-   A9 C0       LDA   #$C0
971E-   8D F3 04    STA   $04F3
9721-   A9 59       LDA   #$59
9723-   8D F5 04    STA   $04F5
9726-   A9 FF       LDA   #$FF
9728-   8D F6 04    STA   $04F6
```

```
; continue the boot
972B-   4C 00 04    JMP   $0400

*BSAVE TRACE2,A$9600,L$12E

*9600G
...reboots slot 6...
...read read read...
<beep>

Success! I have the entire game in
memory.

*2000<800.1FFFM

*C500G
...
]BSAVE PM 0800-1FFF,A$2000,L$1800
]BRUN TRACE2
...reboots slot 6...
<beep>

*C500G
...
]BSAVE PM 2000-7FFF,A$2000,L$6000
]BRUN TRACE2
...reboots slot 6...
<beep>

*2000<8000.BEFFM

*C500G
...
]BSAVE PM 8000-BEFF,A$8000,L$3F00

That's it! Let's load it all into
memory and run it and see what happens.
```

```
]BLOAD PM 0800-1FFF,A$800
]BLOAD PM 2000-7FFF,A$2000
]BLOAD PM 8000-BEFF,A$8000
]CALL -151

*6000G
```

The game starts! And does its fancy speech synthesis thing! And I can get to the main menu. Oh, but wait... from the main menu, when I press "0" to start the game, it crashes:

```
BFF2-    A=FF X=00 Y=32 P=B1 S=E4
*
```

Listing around there certainly highlights the problem...

```
*BFF0L

BFF0-   00          BRK
BFF1-   00          BRK
BFF2-   00          BRK
```

...there's nothing there, that's the problem. But why it is calling $BFF0 or $BFF2, or anything in that range? That wasn't ever initialized; the game data only went up to $BEFF.

I ruminated on this for a while. Then I remembered this as-yet-unexplained jump table that the original disk created during the loading process:

```
; Set up a jump table at $BFF0 (for
; what?)
2400-   A9 85        LDA   #$85
2402-   8D F1 BF     STA   $BFF1
2405-   A9 05        LDA   #$05
2407-   8D F2 BF     STA   $BFF2
240A-   A9 CD        LDA   #$CD
240C-   8D F4 BF     STA   $BFF4
240F-   A9 06        LDA   #$06
2411-   8D F5 BF     STA   $BFF5
2414-   A9 4C        LDA   #$4C
2416-   8D F0 BF     STA   $BFF0
2419-   8D F3 BF     STA   $BFF3
```

Mystery solved. It sets up a jump table during boot because it's actually going to call it later. (I'm a genius; I just disguise it really well.)

```
*BFF0:4C 85 05 4C CD 06

*BFF0L

BFF0-   4C 85 05    JMP   $0585
BFF3-   4C CD 06    JMP   $06CD
```

I don't know what's supposed to be at $0585 or $06CD, but it's certainly not there now. On the original disk, that was where the RWTS lived (on the text page). That's long gone now. But I do have a copy.

```
]PR#5
...
]BLOAD BOOT1 0400-07FF,A$2400
]CALL -151
```

```
*2585L

; I've seen this before -- it's how the
; original disk read the game into
; memory during boot. Zero page $FD
; holds the high byte of the target
; memory address.
2585-   A9 40       LDA   #$40
2587-   85 FD       STA   $FD
2589-   A9 00       LDA   #$00
258B-   85 FC       STA   $FC
258D-   A9 0C       LDA   #$0C
258F-   8D EE 07    STA   $07EE
2592-   AE E4 07    LDX   $07E4

; turn on the drive motor
2595-   BD 89 C0    LDA   $C089,X

; wait loop
2598-   A9 FF       LDA   #$FF
259A-   20 D8 07    JSR   $07D8

; seek to track
259D-   A9 0C       LDA   #$0C
259F-   8D ED 07    STA   $07ED
25A2-   20 CE 06    JSR   $06CE

; more waiting
25A5-   A9 64       LDA   #$64
25A7-   20 D8 07    JSR   $07D8

; read track
25AA-   20 3A 06    JSR   $063A

; error, branch
25AD-   B0 30       BCS   $25DF
```

```
; read another track, but at $4C00 this
; time
25AF-   A9 4C        LDA   #$4C
25B1-   85 FD        STA   $FD
25B3-   A9 0E        LDA   #$0E
25B5-   8D ED 07     STA   $07ED
25B8-   20 CE 06     JSR   $06CE
25BB-   A9 64        LDA   #$64
25BD-   20 D8 07     JSR   $07D8
25C0-   20 3A 06     JSR   $063A
25C3-   B0 1A        BCS   $25DF

; and another track, at $5400
25C5-   A9 54        LDA   #$54
25C7-   85 FD        STA   $FD
25C9-   A9 10        LDA   #$10
25CB-   8D ED 07     STA   $07ED
25CE-   20 CE 06     JSR   $06CE
25D1-   A9 64        LDA   #$64
25D3-   20 D8 07     JSR   $07D8
25D6-   20 3A 06     JSR   $063A
25D9-   B0 04        BCS   $25DF

; turn off drive motor and return
25DB-   BD 88 C0     LDA   $C088,X
25DE-   60           RTS

; try again, from the beginning
25DF-   20 BD 06     JSR   $06BD
25E2-   4C 85 05     JMP   $0585
```

This explains the behavior I saw on the original disk. When you select "Start Game", it accesses the disk, does its fancy speech synthesis thing again, then starts the game. I initially thought this disk access was some sort of secondary copy protection, but I was wrong. This routine is what (re)loads the speech data from disk and stores it in graphics page 2 ($4000..$5FFF). The animated introduction sequence (and the game itself) use that hi-res graphics page, so the game has to reload the speech data from disk whenever it needs it.

(This also explains why other cracks only speak once, on initial load, then never again. They NOP'd out the code that reloads the speech data and calls the speech routine a second time.)

Meanwhile, what's at $06CD?

```
*26CDL

26CD-   60          RTS
```

Seriously, that's it.

~

At this point, I've analyzed the original game enough to reproduce it. Cracks like this are really more like reproductions. I can't retain the original disk's structure because it's not based on 16-sector tracks. But I've captured the entire game and saved it off into a series of files. Now it's time to reassemble those files into a self-booting disk that uses a standard disk structure, i.e. one that is copyable with COPYA.

Still, I want my copy to reproduce the experience of the original game as faithfully as possible. That means switching to the uninitialized hi-res graphics screen as soon as possible when the disk boots, then watching the graphical title screen progressively appear as that memory range is loaded from disk. It also means re-running the fancy speech synthesis thing when you select "Start Game", something that (to my knowledge) no other crack does.

This will require making some design decisions and writing some original code, which, like all programmers, I will be unhappy with. So it goes.

Design decision #1: rather than reload the speech data from disk as needed, I'm going to put it in the language card. Maybe 64K wasn't a reasonable assumption in 1983, but I think it's pretty safe in 2014. (I used a similar technique with Repton -- my very first crack! -- to save and restore the demo and title screen after the game ended.) I can always add a check so that, on a machine with less than 64K, it simply does nothing rather than crash.

Design decision #2: a custom loader. I can't rely on having DOS in memory throughout the game. Even a "64K" DOS like Diversi-DOS uses 256 bytes of main memory ($BF00..$BFFF), and the game overwrites at least six of those for the jump table at $BFF0. And I'll need some place to put my custom code that moves the speech data to and from the language card.

Plus, custom loaders are fun.(*)

(*)not guaranteed, actual fun may vary

Like the original disk, I'll start by showing the (uninitialized) hi-res graphics page. Then I can re-use the disk controller ROM routine to read my custom RWTS into the text page. (The original game also does this.) The RWTS will load the rest of the game into $0800..$BFFF. Then I can jump to $BF00 to copy the speech data into the language card and do whatever other custom initialization I need. ($BFF0..$BFF5 will still be used for the jump table, but instead of jumping to $0585 to reload the speech data, it will call my routine somewhere in $BF00..$BFFF.) Then I can jump to the start of the game at $6000.

It will all happen very quickly.

First, a disk layout. There's plenty of space, but I want to make sure that I document what I'm doing so I don't have to reverse engineer it in six months when I try to reuse this on some other game.

track	sectors	memory range
00	00-03	$0400..$07FF (RWTS)
01	00-07	$0800..$0FFF
02	00-0F	$1000..$1FFF
03	00-0F	$2000..$2FFF
04	00-0F	$3000..$3FFF
05	00-0F	$4000..$4FFF
06	00-0F	$5000..$5FFF
07	00-0F	$6000..$6FFF
08	00-0F	$7000..$7FFF
09	00-0F	$8000..$8FFF
0A	00-0F	$9000..$9FFF
0B	00-0F	$A000..$AFFF
0C	00-0F	$B000..$BFFF

My custom loader went through a few iterations. Early on, I made some simplifications (in retrospect, perhaps oversimplifications) to the RWTS code so that it can only load entire tracks. Since the game is $B800 long, it ends up loading a small range of memory more than once. Specifically, $0800..$0FFF is on track $01, sectors 0-7, but it really loads all of track $01 into $0800..$17FF, then overwrites $1000..$17FF when it loads $1000..$1FFF from track $02. So it goes.

The RWTS is so fast, you won't even notice. Seriously, it's really fast. I didn't write it, although I've adapted it heavily. I originally found it in CompatiBoot, but I've since found variations of it on original disks dating back to 1981. It uses in-place denibblization (like Apple Pascal and later versions of ProDOS). It also uses a "scatter read" function (like Locksmith Fast Disk Copy) that reads whatever sector is under the drive head at the moment.

It's really fast.

Anyway, this is the program I wrote to put the game code onto my disk. It writes to the disk in slot 6, drive 1, and there is no confirmation or error checking.

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

]PR#5
...
]BLOAD PM 0800-1FFF,A$800
]BLOAD PM 2000-7FFF,A$2000
]BLOAD PM 8000-BEFF,A$8000
]CALL -151

; write $0800..$0FFF to track $01,
; sectors $00..$07
0300-   A9 08       LDA   #$08
0302-   85 FF       STA   $FF
0304-   20 30 03    JSR   $0330
```

```
; write $1000..$BFFF to tracks $02..$0C
0307-   A9 02       LDA   #$02
0309-   8D 8C 03    STA   $038C
030C-   A9 00       LDA   #$00
030E-   8D 8D 03    STA   $038D
0311-   A9 10       LDA   #$10
0313-   8D 91 03    STA   $0391
0316-   A9 B0       LDA   #$B0
0318-   85 FF       STA   $FF
031A-   4C 30 03    JMP   $0330
...

; track write subroutine (slow because
; it writes sectors in ascending order,
; but who cares)
0330-   A9 03       LDA   #$03
0332-   A0 88       LDY   #$88
0334-   20 D9 03    JSR   $03D9
0337-   AC 8D 03    LDY   $038D
033A-   C8          INY
033B-   C0 10       CPY   #$10
033D-   D0 05       BNE   $0344
033F-   A0 00       LDY   #$00
0341-   EE 8C 03    INC   $038C
0344-   8C 8D 03    STY   $038D
0347-   EE 91 03    INC   $0391
034A-   C6 FF       DEC   $FF
034C-   D0 E2       BNE   $0330
034E-   60          RTS
...
0388- 01 60 01 00 01 00 FB F7
0390- 00 08 00 00 02 00 00 00

*BSAVE MAKE,A$300,L$98

*300G
...write write write...
```

Now the actual game code is on tracks $01..$0C. The memory range $BF00..$BFFF (written to track $0C, sector $0F) contains the Diversi-DOS 64K stub. I will need to zero that out later in a sector editor and it with my custom game initialization code.

Meanwhile, on track $00, I'll need a bootloader and an RWTS. I wrote all of this in a sector editor (that was fun), but I'll list the code here as if we were reading it in the monitor.

T00,S00:

```
; load sector number (see table below)
0801-   AD D1 08    LDA   $08D1

; out of sectors?
0804-   30 CF       BMI   $07D5

; display hi-res graphics page 1
; (uninitialized)
0806-   2C 50 C0    BIT   $C050
0809-   2C 54 C0    BIT   $C054
080C-   2C 57 C0    BIT   $C057
080F-   2C 52 C0    BIT   $C052

; save sector number
0812-   85 3D       STA   $3D

; high byte of address to load sector
0814-   A9 04       LDA   #$04
0816-   85 27       STA   $27

; increment sector number (above)
0818-   EE 02 08    INC   $0802
```

```
; increment memory address (above)
081B-   EE 15 08    INC   $0815

; set up jump to disk controller ROM
; routine to read sector (X register
; contains slot x 16 at this point)
081E-   8A          TXA
081F-   4A          LSR
0820-   4A          LSR
0821-   4A          LSR
0822-   4A          LSR
0823-   09 C0       ORA   #$C0
0825-   8D 2A 08    STA   $082A

; jump to ROM to read sector (jumps to
; $0801 when done)
0828-   4C 5C 00    JMP   $005C

; minimal physical-to-logical sector
; table for the sectors we care about
; (0..3)
08D1-   09 0B 0D 00

; branch here when done reading sectors
; (this is stored in T00,S00 but read
; into $0700 by the sector read loop,
; so by the time this executes it's
; located at $07D5)

; initialize some RWTS code that relies
; on the slot number x 16 (which is
; in the X register at this point)
07D5-   8A          TXA
07D6-   09 8C       ORA   #$8C
07D8-   8D 8C 04    STA   $048C
07DB-   8D A3 04    STA   $04A3
07DE-   8D B9 04    STA   $04B9
07E1-   8D CD 04    STA   $04CD
07E4-   8D E2 04    STA   $04E2
```

```
; zero page $FF holds the current track
; number (the actual number, not x2)
07E7-   A0 00       LDY   #$00
07E9-   84 FF       STY   $FF

; zero page $FA holds the track number
; we want to read
07EB-   C8          INY
07EC-   84 FA       STY   $FA

; Y register also holds the number of
; tracks to read
; Accumulator holds high byte of the
; address to store track data
; X register must be slot number x 16
; on entry (it already is, so no
; initialization needed)
07EE-   A9 08       LDA   #$08

; read entire track into consecutive
; memory ($0800..$17FF)
07F0-   20 00 04    JSR   $0400

; now read $0B tracks into $1000 and up
; (this will fill the rest of main
; memory)
; note: zero page $FA is automatically
; incremented after reading a track, so
; this second call starts reading at
; track 2
07F3-   A0 0B       LDY   #$0B
07F5-   A9 10       LDA   #$10
07F7-   20 00 04    JSR   $0400

; turn off drive motor (X register is
; always slot number x 16 when RWTS
; returns)
07FA-   BD 88 C0    LDA   $C088,X
```

```
; jump to my game initialization code
07FD-   4C 00 BF    JMP   $BF00
```

Sectors 1-3 on track 0 contain the RWTS that I spoke of earlier. (Sector 3 is loaded at $0400, sector 2 at $0500, and sector 1 at $0600. Sector 0 is reloaded at $0700, which is why the weird backwards branch at $0804 works.)

As you can see from the boot0 code listing, the RWTS has only one entry point, at $0400. It reads a given number of tracks into consecutive memory ranges. The RWTS assumes the drive motor is already on, and it never turns it on or off.

Meanwhile, T0C,S0F holds my custom game initialization routine. $07FD jumps here as soon as the entire game is loaded into memory.

```
; initialize keyboard/video/screen
; (note: this also wipes the RWTS from
; memory, since it's on the text page)
BF00-   20 89 FE    JSR   $FE89
BF03-   20 93 FE    JSR   $FE93
BF06-   20 58 FC    JSR   $FC58
```

```
; check for language card
BF09-   AD 83 C0    LDA   $C083
BF0C-   AD 83 C0    LDA   $C083
BF0F-   A9 AA       LDA   #$AA
BF11-   8D 00 D0    STA   $D000
BF14-   4D 00 D0    EOR   $D000
BF17-   D0 35       BNE   $BF4E
BF19-   4E 00 D0    LSR   $D000
BF1C-   A9 55       LDA   #$55
BF1E-   4D 00 D0    EOR   $D000
BF21-   D0 2B       BNE   $BF4E

; yes we have a language card (a.k.a.
; at least 64K), so copy the speech
; data into it so we can restore it
; later
BF23-   A2 20       LDX   #$20
BF25-   A0 00       LDY   #$00
BF27-   B9 00 40    LDA   $4000,Y
BF2A-   99 00 D0    STA   $D000,Y
BF2D-   C8          INY
BF2E-   D0 F7       BNE   $BF27
BF30-   EE 29 BF    INC   $BF29
BF33-   EE 2C BF    INC   $BF2C
BF36-   CA          DEX
BF37-   D0 EE       BNE   $BF27

; switch back to ROM
BF39-   AD 82 C0    LDA   $C082

; set up a hacker-friendly reset vector
BF3C-   A9 5A       LDA   #$5A
BF3E-   8D F2 03    STA   $03F2
BF41-   A9 BF       LDA   #$BF
BF43-   8D F3 03    STA   $03F3
BF46-   49 A5       EOR   #$A5
BF48-   8D F4 03    STA   $03F4

; start the game
BF4B-   4C 00 60    JMP   $6000
```

```
; no language card --
; disable subroutine that would restore
; speech data from the language card
BF4E-   A9 60       LDA   #$60
BF50-   8D F0 BF    STA   $BFF0

; disable call to play speech
BF53-   A9 2C       LDA   #$2C
BF55-   8D E4 8E    STA   $8EE4

; continue with initialization (above)
BF58-   D0 DF       BNE   $BF39

; hacker-friendly reset vector --
; clear screen and jump to monitor
BF5A-   20 58 FC    JSR   $FC58
BF5D-   20 2F FB    JSR   $FB2F
BF60-   4C 69 FF    JMP   $FF69

; copy speech data back from language
; card into main memory
BF74-   A9 D0       LDA   #$D0
BF76-   8D 87 BF    STA   $BF87
BF79-   A9 40       LDA   #$40
BF7B-   8D 8A BF    STA   $BF8A
BF7E-   AD 80 C0    LDA   $C080
BF81-   A2 20       LDX   #$20
BF83-   A0 00       LDY   #$00
BF85-   B9 00 D0    LDA   $D000,Y
BF88-   99 00 40    STA   $4000,Y
BF8B-   C8          INY
BF8C-   D0 F7       BNE   $BF85
BF8E-   EE 87 BF    INC   $BF87
BF91-   EE 8A BF    INC   $BF8A
BF94-   CA          DEX
BF95-   D0 EE       BNE   $BF85
BF97-   AD 82 C0    LDA   $C082
BF9A-   60          RTS
```

And don't forget the jump table.

```
; jump to routine that re-loads the
; speech data (from the language card
; instead of from disk)
BFF0-   4C 74 BF    JMP   $BF74

; do nothing (this originally jumped to
; an "RTS" on the text page, so we're
; not losing any functionality here)
BFF3-   60          RTS
```

With that jump table in place, I don't need to make any modifications to the game code ($0800..$BEFF). As far as the game can tell, it's running in its original environment.

Quod erat liberandum.

EOF