JOTD HD Install Collection

COLLABORATORS			
	TITLE : JOTD HD Install Collection		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		August 8, 2022	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

1 JOTD HD Install Collection **1**

1.1 JOTD HD Install Collection . . . 1
1.2 Introduction . . . 1
1.3 How to write installers and patches . . . 2
1.4 Problems . . . 3
1.5 To do . . . 3
1.6 Disk File Makers . . . 3
1.7 Save your time . . . 4
1.8 Coders' Home . . . 4
1.9 Credits . . . 10
1.10 Legal part . . . 12
1.11 Contact me . . . 12
1.12 Degrader-part features . . . 13
1.13 Crashes . . . 14
1.14 Disk2File . . . 14
1.15 Boot2File . . . 15
1.16 Rob2File . . . 16
1.17 RipPsyFiles . . . 17
1.18 Grem2File . . . 18
1.19 PsyDir . . . 19
1.20 The Cadaver case . . . 19
1.21 Quitting games . . . 20
1.22 Hardware used . . . 21
1.23 Software used . . . 21
1.24 Documentation . . . 22
1.25 Patching . . . 22
1.26 It's up to you Mulder . . . 23
1.27 HD Installing/Loading techniques . . . 24
1.28 Dumb Blitter Coding . . . 25
1.29 CPU prefetch problem . . . 26

1.30 24bit addresses . . . 27
1.31 Fastmem at C00000 . . . 28
1.32 1 Megabyte of chipmem . . . 29
1.33 CPU is faster than a 68000/7MHz . . . 30
1.34 STACK FRAME . . . 30
1.35 UNHANDLED INTERRUPTS . . . 31
1.36 Freezes (on keypresses) . . . 31
1.37 WEIRD ROM ACCESSES . . . 32
1.38 MOVEP . . . 32
1.39 CacheControl problems . . . 33
1.40 The floppies holding files . . . 34
1.41 Writing installation program/script . . . 35
1.42 Starting the game . . . 37
1.43 Loading virtual tracks . . . 39
1.44 Loading virtual files . . . 41
1.45 Patching disk changes . . . 41
1.46 Configuring extension memory . . . 42
1.47 Handling supervisor mode . . . 43
1.48 Removing protections . . . 44
1.49 Switching drive led off . . . 44
1.50 Removing DSKRDY code . . . 44
1.51 Managing with crunched code . . . 45

Chapter 1

JOTD HD Install Collection

1.1 JOTD HD Install Collection

```
 ************************************************************************


      The JOTD HD-Installer Guide
A set of programs to patch/install your games on HD
   © Copyright 1995-99 Jean-François Fabre

************************************************************************
```

Introduction

Problems

To do

How to...

Legal part

Credits

Contact me

1.2 Introduction

```
When the Amiga appeared on the market, it had only floppy drives, and people were ↩
    very
happy with it. But progressively, amiga users got hard drives and that is really a ↩
     pity
that the games continued to be loaded from floppy exclusively, due to a copy  ↩
    protection
problem, while the PC versions of the same games were HD installable, with a  ↩
    password-style
```

protection or a keydisk check instead.
Some adventure games were of course HD installable on the Amiga, but the best beat ↩
-em-ups
and shoot-em-ups were not installable at all, and even worse: the disks were in a ↩
very
special nibble format, so no way to read the data on the disk directly!
Some games do not take care of the memory expansion cards and spend their time ↩
loading...

I think this contributed to the death of the Amiga, though now all the AGA games ↩
run from
HD without problems (XTR, XP8, Worms...)

I tested some HD installers which did not work but that gave me the idea to write ↩
my own ones.

I also experimented problems with some games that did not work on CPUs higher than ↩
68000, or
because of a faulty memory detection, or several other reasons
(2Mb chipmem causes harm sometimes).

So, I wrote patches too, and HD installers with patches.

In this guide, I wanted to explain how my loaders work and how to configure them, ↩
but
also how to code your own installers or how to rip data from any disk.
I also explain why some old games crash or behave weird on newer machines.

1.3 How to write installers and patches

Some people asked me how hard it was to write installers and ↩
patches, and which were
my tools to do such programs.

MY TOOLS:

As you already suspected, I use a lot of software, hardware and docs.

Hardware

Software

Documentation
THE THINGS YOU NEED TO KNOW

Patching

HD Loading

Disk file makers

Save your time

Coders home

1.4 Problems

You may experience problems with the installers or the loaders.

Degrader-part features

The game crashes...

The Disk2File program

The Cadaver HD Loader and patcher

Quitting the game

1.5 To do

Some things are left undone:

- Patch a full box of floppies waiting at home.
- Install a quit option on EVERY game.
- Improve the documentation and the programmers autodocs.

Well, all this takes time.

1.6 Disk File Makers

As I explain
here
, some games have got a same disk format
in common. Team 17, Probe and Psygnosis use a standard non-copiable format
I used to copy the same code in many of my installers but I now decided it was ↩
worth
writing an in-line command to handle all of the disks, which has many advantages:

- Possibility to include the commands in Commodore/Escom Installer scripts
- More flexible installers. You can install one particular disk 'by hand'
- You can try those programs on the games you'd like to see on HD to see if they succeed. If they do, warn me and I'll create a loader very easily (or write it yourself if you can)

If you are a hd-installer programmer, you can use my programs and include them in your archive provided you e-mail me about that and you tell in you documentation that I wrote the ripper program (advertising, always...)

```
                Disk2File

                Boot2File

                Rob2File

                Grem2File

                RipPsyFiles

                PsyDir
                  If the rip succeeds for a game I did not write an install for,  ↩
                     please e-mail me to
   warn me about it, and I'll be able to write an installer and a loader very  ↩
      quickly.
```

1.7 Save your time

```
When you want to patch/hd install a game that comes in NONDOS disk, first check  ↩
   the
company/team who did it. That can be important, because you can sometimes  ↩
   partially
or completely reuse some disk rippers that already exist.

- If it's dostracks (with or without copylock), use disk2file or DIC.

Now if the disk has got most non dos tracks (X-Copy red sectors):

- If it's a Team 17 game use Rob2File (first find the decode key)
- If it's a Probe game, try Rob2File with key=0
- If it's a Psygnosis game, try rippsydir. If a dirfile is created, you can say  ↩
   bingo.
- If it's a Gremlins game, use grem2file. That always works.

Another little example: Harry  wrote a disk imager for Agony (Psygnosis)
for a loader I made. I noticed that the programmers were the same as the ones who
coded Unreal (from Ubi Soft). I tried the imager on Unreal and it worked, because
the disk format had been developped by the programmers and they used it in both  ↩
   games,
even if they worked for another publisher.

Most of the time, disassembling the loader routine and checking the syncs is  ↩
   enough
to tell if ou already saw a imager for this kind of track.
```

1.8 Coders' Home

```
In this paragraph, I included some discussions we had by e-mail with Harry, Mr
Larmer, Abaddon and me.
```

I think it's interesting, and it could be a kind of small FAQ.

1. How to insert a quit option in a game? (Jeff->Abaddon)

This example follows JOTD HD Startup syntax:

The quit option is very easy anyway once you located the keyboard
routine. There's normally a:

```
        move.b $bfec01,d0
        not.b d0  (or eor.b #$ff,d0)
        ror.b #1,d0

        and in d0 you've got the rawkey code of the key
        (add $80 if the key is released)

        e.g: $44 means return, $59 means f10, $5f means HELP...

        for instance patch the move.b $bfec01,d0 location (6 bytes)
by a PATCHUSRJSR or PATCHABSJSR KbInt

KbInt:
        move.b $bfec01,d0   (like the original, because you stole 6 bytes)
        move.l d0,-(a7)
        not.b d0  (or eor.b #$ff,d0)
        ror.b #1,d0
        cmp.b #$59,d0   ; f10 pressed?
        bne     noquit
        JSRGEN  InGameExit  ; will not quit if not enough mem
noquit:
        move.l (a7)+,d0
        rts
```

If the value of D0 is stored by the game
(move.b d0,$45068) you can patch this location

that avoids to save d0 and make the not.b and ror.b.

2. SWIV: Gfx status bar error (Harry->Mr Larmer)

The hiscore/lives bar did not appear properly in SWIV.

First: Boot, freeze and look on the picture. OOPS - nothing to see from a
scoredisplay. Hm... Checking sprites... No, sprites are used for shoots.
Looking at the copperlist... Ah! while displaying the status bar the game
switches in 5-plane-mode. Now i tried to find the part of program which sets
$dff1a0-$dff1c0. After 2 h i abandoned and tested something: I turned off all
the interrupts, set the sr to 2700 and set the pc on a loop 1. bra .1
And then i got a very negative surprise: The scoredisplay is done by
modifying only color $10! I dont know WHY it works at all (and color $11-$1F
isnt also used), but since it works i had to program an aga-workaround for
this. On AGA, i duplicated the score-plane as plane 6 and set color $30-$40.
Sorry, but i wasnt able to fix that colorbar too without BIG recodings of
the game. That fix took already 7 h.

3. Vroom and Wings of Death fixes (Harry -> Mr Larmer)

Ok, ill explain how i search bad code on some examples.
Preface: There is no best method to search bad code. The best method is
that where you reach your goal quickly. But i have now my method to search.

Wings of Death: Sound FX error.
First thought: Sound is played in the vbi (Vertical Blank Interrupt) and
hippel has an soundplayer-module. So i looked for a far call in the vbi.
Ok, i found the soundmodule. Then i had to find the entry in the jumptable
where an effect is played. just put breakpoints in it, delete the breakpoints
which are reached while nothing special happens (DO NOT shoot enemies while
this time, it makes boom-sounds!). Then shoot one enemy and the appropriate
breakpoint is reached. Now you have the routine. Look what it does and what
could be wrong, there is no general method from here.

Vroom: Gfx error.
First thought: Its not the disk reader and not the decruncher, because the
program itself and the soundplaying seems to be correct.
So i looked where the first screen was. Then i rebooted the game and put
a memwatchpoint there. Ok, the memwatchpoint was reached and the instruction
just executed was the one i looked for (always verify this! If a routine is
executed in the interrupt, the memwatchpoint-method fails that way).
Ok, now you have the routine. Look on it. But its such a spaghetti-code that
i decided to reboot the game and put a breakpoint at the beginning.
Then i traced it, with a list of sources for faults in mind. After a while
i found that prefetch-bug. Now search for similar instructions in the
other routines, since programmers rarely change their method to program
something.

4. Jeff -> some guy who thinks he can teach me how to behave

xxx wrote:

> Why do you have to be differant and have people start
> registering your software?
>
Because I need some more motivation for keeping patching games.
Its' sometime a lot of fun, but when it comes to adapt the
54636th version of a game I've already got, it becomes a bit
annoying.

> You don't have to register WHLoad. No-one who ever
> made a patch except the shitty Air-Soft people have ever
> asked to register their patches.
>
The difference between AirSoft productions and mine is that you
can continue to use JST limited if you don't agree with the
shareware policy (though I think that 10$ is not really a lot
to be able to use more than 100 installs available for JST,
and more to come).
JST limited will continue to be updated and debugged owing to
the people who will register it. Only them will get the new features
but JST is better than the old link method of my old installs, and
then it's good for every people, even those who don't consider that
it's important to register.

When you say that you don't need regisitering WHDLoad, the future
will tell us if you're right or wrong (Bert will understand this one).

> Yes, you have done alot of work to create the patch system,
> but you did that on your own. Who asked you to do it?

I could forward every mail request to you your mailbox will burst.

> The Amiga is dead, why would anyone want to spend anymore
> money on it?

If you think this way, buy a fuckin' peecee.

 Also, what is this crap about not working
> with pirated copies?

that's an interesting point, and should be part of a FAQ.

>
> #1, Name one company that you have patched that still exists
> today. They made their software (even when there were
> hard drives readily and affordable to the Amiga community)
> on un-copiable disks. In my book they deserve to be pirated.
> I payed good money for software that I cannot use or backup.
>
First, if you have ever bought an original game (I'm quite joking)
you'd know that the disk format is different most of the time.
The sectors are non-dos and non copiable except with X-Copy
or a hardware copier.
Pirated copies of dos copiable games are mostly compatible with
my loaders. And I make no effort to prevent cracked copies users
to use them (except for some peculiar cases (e.g Chaos Engine 2))

So in most of the cases, supporting a pirated version means
an extra work for me, and I'd loose respect from some people
other than you!
And I don't want to spend hours removing stupid code from
the crackers.

> #2, Do you really want us to think that YOU did not keep a
> working copy of those games that you patched? I mean the
> ones where someone sent you an original disk.
>
What if there's a problem with the game, or update?

> Yea, you deleted it right after you sent it back to the owner.
>
ok, you won. you make me sick.

> anyway, You lost my respect.

You never had mine.

Jeff

5. Zeleny & Troda -> Jeff

Scandoubler sync problem

When i changed my oldie C=1084S to scandoubler for CV3D card (Phase5) & VGA monitor, i found out that many, especialy new demos/games producing trash display. Because all OS screens and majority of other things don't do that, then i suspect any bug in copperlist. After asking for help my friend Zeleny, he provided quickly solution - problem is in COLOR bit in BPLCON0. It MUST be always SET !!! Otherwise there will be problem with synchonisation on all Amigas with scandoubler except PicassoIV FliFi because this works as a grabing device, istead of all others ones, which only double vertical frequency.

This is the article wrote especially to all authors of all demos.
You should check your production and correct it.

This bug is in following demos:
Ride/Jamie Skarla, Relic/Nerve Axis, EraserHead/Maq, Dreizehn/Oxyron,
Goa/TBL, Manipulations by Venus Art, and many others...

Because scandoubler is nowadays too widely spread equipment and in future it's number will be growing, please TAKE CARE at this annoying bug.

If you are still not persuaded, watch the demo Ride on an Amiga with scandoubler !!!
(we fixed it in hex editor and now it really works perfectly !!! ;-)))))

by Zeleny & Troda in co-op

6. Jeff -> Codetapper

On 14-Déc-99, you wrote:

> Hi there
>
> I was wondering if you have any "how to deprotect" type documents for
> RNC protection you could possibly give me or point me to where I get
> them.
>
> The one I mean is the standard one where you put a routine at $10
> and then do an illegal instruction (which WHDLoad doesn't work with and
> I assume JST aswell) followed by moving a routine onto the stack (trace
> function) and then installing that at $24 followed by a whole bunch of
> junk which is encrypted.
>
right.

> Does this routine EVER modify memory used by the game to your knowledge,
> or it simply reads data from the disk and generates the key value in d0
> which is used SOMEWHERE else in the game?
>
It sometimes changes some zero page locations! The trick is to dump chipmem to disk before calling copylock, calling copylock with the original disk (!) in the drive, try to breakpoint in the end of the copylock, dump

chipmem, and then compare both. Try to track down isolated values that changed, and try to see where the
address is used. E.g: Chaos Engine: $100 is changed, and checked somewhere in the game, Lemmings: lots of locations, Assassin: $FC changed, and crashes when timer reaches 2:00 (the suckers!! just find the clock and you can see the
memory $FC test!!!)

> The reason I ask is I am working on "Rugby: The World Cup"/Domark
> (supplied by Carlo Pirri - bless his heart!) and I thought I'd compare the
> game to the Crystal crack I have to get the key.
>
> But they modify code inside the $10 illegal routine rather than the usual
> way I've seen in most games:
>
>   pea   crypt(pc)
>   move.l (sp)+,$10
>   illegal
> crypt ...
>
> and they replace with:
>
>   move.l   #$12345678,d0
>   bra   xxxxxxx
>   ...
> xxxxxx jmp $9280
>
> ie. Crystal have changed about 8-10 bytes well down the crypt routine, so
> it's still running the illegal code and crashing WHDLoad (and I assume JST
> if I did the loader in that).
>
Other examples: Addams Family, MK 2, where the pirates changed a few bytes inside the crypted routine (how did they do that) and the copy lock is disabled,
that's rather tricky but it works. I've got to test Addams Family and MK2 again
but I'm pretty sure that it works OK with JST, even though ILLEGAL is called.

> Have you got a method/technique for these? Has anyone written a utility to
> simulate the RNC protection in any way? ie. Decode all the illegal junk
> and replace the whole routine with a normal one which can run in
> supervisor mode?
>
> Also I have a 2nd 1200 with an AR1200 cartridge and this piece of junk
> can't trace through the RNC protection. As soon as I hit the illegal
> instruction, bang - guru time. From your docs and stuff it sounds
> like you also have this cartridge, have you any tips for this?
>
I've got only A500 version. But you should put a breakpoint at the address in $10 instead of tracing and make a x. Less stack problems.

> One final thing (sorry this is so long!) is it true the only way to decode
> these is basically with a cartridge on an amiga 500 and you break the
> routine right near the end and intercept the key in d0 (after the illegal
> junk has returned)? I was told by Bored Seal this is what the old crackers
> used to do, is this true to your knowledge?

```
>
yep. When it's possible I do this. I had once to decrypt a routine (Fire &
Ice) and I don't remember how I did exacty (it was tricky!).
Roughly done in 2 passes:

First locate the routine called after a trace (in $24) and store the current
decrypted PC of the copylock routine in some address, leave the copylock
running, and get the value in the address in the end.

Second, change the $24 trace routine in a way that it jumps to a location
when the PC of the copylock routine reaches the value you got above. In
this location you put a breakpoint when reached. You've got then the end of
the copylock routine, and you know what the program does then (that's
because sometimes
the copylock does not end by RTS but by a JMP somewhere and it's
encrypted!).

Once you got the end of the copylock, try to see what the program does, and
save necessary decrypted code that the program will call next (usually a
short loader routine).
Then in your loader, call directly the decrypted routine that you included
in the loader code, and that you copy directly where it should be, no need
for copylock.

All this assumes that you've got the original disk (or a cyclone copy).

You can also try the Copylock decoder from Mr Larmer (I'll provide you a
copy if you don't have it).


Bye

Jff
```

1.9 Credits

```
In this page, I would like to list the people who contributed to this rich  ↩
    installer
collection (me, among the others :-) )

HD Installers and patches coding:        Jean-François Fabre (of course :-))
                                        Ralf Huvendiek (also uses my JST code)
                                        Keith Krellwitz (also uses my JST code)
                                        Harry (hints and some disk imagers +  ↩
                                            WHDLoad stuff)
                                        Mr Larmer (hints, hacks, WHDLoad stuff)
                                        Codetapper
                                        Dr Zarkov
                                        Derek Noonburg (original Marble Madness  ↩
                                            patch)
                                        Stefan Boberg (author of a version of  ↩
                                            AlienBreed2 HD)
                                        Kenneth C. Nilsen (workbench/cli startup  ↩
                                            asm code)
                                        Bert Jahn (installer scripts and CIA code)
```

Alain Malek (for the CIA store/restore code)
Joël Patthey (rip part of the Cannon Fodder patch)

Various trainers, hints and hacks:

me (again)
Crystal
Ralf
Harry
Prestige
Legend
Felon
JOKER
Abaddon
Bert Jahn
John Girvin
The Band
Skid Row

Original games suppliers:

Gerald Yuen
Benjamin Brandt
Jossa Andersson (send me a mail please, I lost yours)
Chris Gregory
Paul Dossett
Chris Coulson
Bill Bennett
Keith Krellwitz
Dave Clarke
Timo Kaikumaa
Troy Pladson
Martin Coleman
Walter Gierholz
Keron Wilkinson
Ray Niblock
Tom Kajaslampi
Joël Patthey
Mark Kelkermans
Glyn Curtis
Dominic Cresswell
Chris Vella
Sébastien Meyer
Thierry Sillis
Stavros Themistoucleos
and some others I forgot. Thank you

Cheats supplied by:

Keith Krellwitz (SweetCheater)
Brice Allebrand

Other thanks:

- Registered users of JST. I'm not ready to stop my work. THANK YOU!!!!
- Aminet organization for their work, and the (almost) free CDs :-)
- GNU organization. Free and very good stuff, even on the amiga

```
- All cool guys who ported UN*X stuff on the amiga
- N.O.M.A.D for TUDE
- Alain Malek for HRTMon
- Bert Jahn for WHDLoad and his cool installers
- Keith Krellwitz for having adopted JST as a HD startup code for his loaders.
- John Girvin for his installers and e-mails discussions
- Mr Larmer for his installers and useful little proggies ;-)
- Harry for continuous e-mail, support, and cool work
- Frank Wille for Phxass (the best in-line assembler)
- Tim Ruehsen for IRA (the best freeware resourcer)
- Pandora for AGA.guide (the complete guide of custom registers, including AGA)
- B. Schmidt for UAE and S. Devulder for the Amiga version.
- Duchet computers for supporting the Amiga


NO THANKS to:                                   France Festival Distribution. You  ↩
    motherf***er(s).

                                                Crimewatch UK and Mars bar: your  ↩
                                                   installers suck shit, don't work
                                                unless you've got a 68000 with fastmem at  ↩
                                                   $C00000 and are made
                                                for very special cracked versions that  ↩
                                                   nobody's got except you.
                                                Your mackHD.library is crap too. F**k you, ↩
                                                    lamers!

                                                Bill Gates (He'll understand why).
```

1.10 Legal part

```
1. I cannot be responsible for the misuse of my programs in any way
You're not gifted if you manage to lose data with my programs. If you I.Q. is  ↩
   below your
anal temperature (in \textdegree{}C), backup your data NOW. (The I.Q. joke is by  ↩
   Pierre Desproges)

2. As my installers often remove copy-protections, the backups you'll make from  ↩
   the installed
games are for personnal use only. Please do not spread them or upload them on  ↩
   pirate sites.

3. Legal parts are boring. Thanks for reading them.
```

1.11 Contact me

```
If you wish to contact me for any reason or you want to send me original disks for ↩
    me to patch
them, or you want to send me money or Sandra Bullock clones (Sharon Stone copies  ↩
   accepted)
or even the cops, feel free to e-mail me or snail-mail me:
```

My address:

E-MAIL:

jffabre@free.fr

SNAIL-MAIL:

Jean-François Fabre
19 Rue Emile Duploye
34500 BEZIERS
FRANCE

Please send me an e-mail before sending disks as I may already have them.

HTTP:

Visit my patch page, where all my HD work is gathered, along with some other quality installers (all for original software), and some other interesting freeware stuff:

http://jffabre.free.fr/amiga/patches.html

And my home page (totally stupid stuff):

http://perso.club-internet.fr/jffabre/ (english)
http://perso.club-internet.fr/jffabre/index_fr.html (français)

1.12 Degrader-part features

All my HD-Loaders degrade everything they can. They:

- Set VBR to zero for 68020+ CPUs
- Set the proper caches (up to 68060 supported)
- include DTack (built-in)
- Change the display and the sprite resolution to PAL or NTSC (tooltype)
- Allocate memory from the top (MEMF_REVERSE) when possible (OS 2.04+)
- Remove all f***ing disk protections and passwords
- Use fast memory when possible, even if the game does not (Cannon Fodder 2).
- Recognizes different versions of some games when possible.
- Free all the memory if something goes wrong before the startup (a file missing).
- Create no enforcer hits (but the games do :-)) except for some low level code (in this case it's on purpose). But if you run the loader without the data files (the game won't start) , you'll see there are no enforcer hits.

All that means that you normally won't have to use (excellent) programs like KillAGA, TUDE, Setchiprev, Degrader, runlame, etc... to boot the game properly.

1.13 Crashes

```
Some crashes were reported. If you get crashes, the cause may be in this list:

- Too few chip memory, or an absolute memory block allocated by the workbench.
- A program using interrupts in the background.
- FastExec, and some valuable patch programs that move system structures in ↩
    fastmem.
- Enforcer. Always turn it off when you run a game (too bad Cyberguard cannot be ↩
    removed)
- MMU Softkicked amigas may experience problems too because of MMU. Sorry, but I ↩
    think
  that almost no game runs properly with MMU active, even from floppy.
- You did not run setpatch (for 68040/060).
- The version of the original game you installed is different from mine.
- You use a cracked version (it works sometimes, though, with installers using ↩
    disk2file,
  or dos files, ripped or original).
- You're trying to run my programs on a f***ing PeeCee :-)

Before e-mailing me when you can't start the game, try booting without startup- ↩
    sequence
(it's better to run setpatch, though).

Currently, the games able to return to the OS by pressing F10/ESC don't crash, but ↩
     display an error
message and can even create log files. Much better like this, isn't it?
```

1.14 Disk2File

```
                It was a shitty utility I've written (quickly) to create files ↩
                   from trackload disks.
Some other programs are available on aminet. They are a bit better, but I recently
improved disk2file a lot (V1.2); It's now safer and more reliable.
However, Disk2file does not handle read errors (we have to ignore some), so be ↩
    careful,
and check the disk before creating an image (with XCopy or SuperDuper) to see the ↩
    disk
structure. If almost all the sectors are reported faulty, disk2file cannot rip any ↩
     data
from them.

Sometimes this executable was missing in my archives. I'm sorry for that, but it ↩
    does
nothing that other diskimages won't do!
You can find several disk imagers on aminet. One can be found in WHDLoad, one of ↩
    my direct
concurrents' excellent HD loader package. It's called DIC (Disk Image Converter) ↩
    and seems to
be safe to use. Another one is also on aminet in disk/misc but I don't remember ↩
    its name.

NOTE: disk2file (and neither the others disk imagers) cannot be used to create ↩
    disk images of
```

```
copy protected games (except when only one sector is faulty). In this case, a  ↩
   special
installation program is provided in the package (the REAL hard work) or you can  ↩
   also use

                other disk rippers
                I've written in some special cases.

USAGE:

Disk2File can be used as follows:

disk2file <drive unit> <diskimage filename> [SKIPROB]

e.g: disk2file 2 speedball2.adf SKIPROB  will create a diskfile from unit DF2:
in the current directory. SKIPROB causes that Track 0 Head 1 (track 1) will be
skipped, which does not mean that the following offsets will be shifted.
SKIPROB allows not to read a Rob Northen copylock track.

Omitting SKIPROB is not serious in any case, but switching it on on non-protected
disks can be a disaster, since there may be data on the track!

For OS2.0+, the drive is inhibited during the load, which avoid problems.
You can break disk2file at any time using CTRL-C. There were crashes with V1.0.  ↩
   This is now
fixed.
This utility allows to create disk images in a standard raw format, useable by HFM ↩
    Mounter,
or UAE amiga emulator.

This program does not detect errors on non-DOS or protected tracks. Be careful.

The SKIPROB option allows to skip Track 0 Head 1 when the game is copy-protected
using widespread Rob Northen copylock. A zero block will be written.
```

1.15 Boot2File

```
This utility is a cut-down version of disk2file for tracks 0 and 1.
It's useful when you want to rip the bootblock of a disk to see the startup code.
Note that this code begins at offset $C of the bootblock and the only bytes
loaded by the system at startup are the $400 first ones. But in the bootblock
($400 bytes - 2 sectors), the game can call trackdisk.device to load more sectors,
and those sectors are often included in the 2 first tracks of the disk.

This program does not detect errors on non-DOS or protected tracks.


Boot2File can be used as follows:

boot2file <drive unit> <diskimage filename>
```

1.16 Rob2File

```
                Many of the games from Team17 or Probe use the same disk format. I ↩
                    decided to write a general
purpose utility to cover all the cases. Rob2File was born.

You can run rob2file with the following arguments (I did not make any ReadArgs(),  ↩
    sorry):

Arg1 (compulsory): unit number (from 0 to 3) -> Source DF0: to DF3:
Arg2 (compulsory): destination file name
Arg3 (optionnal) : start track (from 0 to 159). Default 0
Arg4 (optionnal) : end track   (from 0 to 159, but at least equal to start track). ↩
     Default 159
Arg5 (optionnal) : disk decode key (hexadecimal).
Arg6 (optionnal) : If you type the keyword OFFSET, the disk file will be created  ↩
    with an offset.

ABOUT THE OFFSET:

IF the game comes in more than 1 disk, disk 1 has got 1 or 2 dos readable tracks
(readable by Boot2File, DMS, or any disk editor). So the Rob format starts at  ↩
    track 2
(e.g Project-X SE), but for other disks, the format starts at track 0 (to avoid to ↩
     lose space).
In your loading routine, you'll have to substract 12*2 sectors if you read from  ↩
    disk 1, which
is a bit annoying (most of all if you forget to do it :-) ).
The offset command allows you to leave bytes to zero (start track * $1800), so the ↩
     disk read command
will match the offset.

E.g:

rob2file 0 projectX.disk1 2 159 123898 OFFSET

will create a file with 2*$1800 (=$3000) bytes to zero, the normal dos tracks.

Conversely, you'll create the diskfile of disk 2 by the command:

rob2file 0 projectX.disk2 0 159 123898

since disk2 holds data even in tracks 0 and 1.

And the files will be exactly the same size (S12_DISK_LEN, see libs.i)

OFFSET can be omitted when you start at track 0 :-)


Sometimes (Mortal Kombat 1 and 2) track 1 is non-dos, but you cannot read it with  ↩
    rob2file.
It is a special protection track (copylock) holding generally no data. You'll have ↩
     to remove
the protection or leave the disk inserted during the game.
My HD-Installers usually remove this protection.
```

```
Mortal Kombat (1 & 2) disks hold data only from track 2, even with data disks, so ↩
    you can
omit OFFSET keyword (I do in my install-script) and then systematically substract ↩
    12*2 sectors
to the sector offset in the Rob Northern loading routine.

ABOUT DISKKEY:

For Probe games, leave to 0, else you'll have to disassemble the game
                Rob loading routines
                to see the key (present in register D4 at rob disk interface). ↩
                   Default is 0 in rob2file.
First try with 0 and start to hack only if it fails.

Some key and track values:

Project-X special edition: 3 disks: 0x123898
Mortal Kombat I & II: all disks: 0x0
Alien Breed II AGA: disk1: 0x123111, disk2: 0x123222, disk3: 0x123333
Body Blows: disk1: 0x13246679, disk2: 0x13246678, disk3: 0x13246677

When the keys are different, it's easier for you when you write a hd loader
to see which disk the game needs (diskchanges are sometimes hard to simulate) :-)


Rob2File V1.0 - The RN Disks image maker by Jean-François Fabre

Usage : rob2file <unit number> <filename> [<starttrack>] [<endtrack>] [<diskkey>] ↩
    [OFFSET]
Defaults are:
  Unit: 0
  DiskKey: 0x00000000
  StartTrack: 0
  EndTrack: 159
```

1.17 RipPsyFiles

```
                I first wrote an installer for Leander, and I wanted to rip the ↩
                   whole disk like with rob2file
but it was harder to patch for a hd loader, so I re-sourced their loading routine ↩
    (given as
an example in the Ripped drawer) and I noticed there were a rudimentary file ↩
    system on the
disk, and so I could rip files and patch the routine at a higher level. That's ↩
    what I did
and I was very pleased when I tried my ripper on Shadow Of The Beast 3 and that it ↩
     (almost)
worked. I think that this filesystem and this disk format is spread in Psygnosis ↩
    productions.
Unfortunately, Awesome, Agony and Killing Game Show use another format, so I have ↩
    to redo
everything from the start with those games :-(

I had the idea to create a file ripper.
```

```
So you can test your Psygnosis games to see if they are rippable and I could write ↩
    clean installer
scripts instead of hard-coded in-line ugly installers.
Code re-use is my obsession.

It's much simpler to use than rob2file:

rippsyfiles <disk unit> <destination directory> [<disk id>]

disk unit: 0 to 3.
dest dir : the directory where the ripped files will be stored
disk id  : not compulsory. The
                psydir
                 command allows you to read the directory.
At offset $10 of the file created, note down the 4 characters of the filename.
It's the disk id.
It's safer to specify the id in installers, but if you want to rip any disk, you  ↩
   can omit it.

You can specify the ID in hexadecimal: 56362A31 or in ASCII between quotes: 'BULK'

For the moment, this utility works only with Shadow Of The Beast 3, Leander,
and the Lemmings series.
Report me any other successes.

I recently updated this tool so reads are system-friendly and there won't be any
DSKRDY problems. Moreover, the pointer will not freeze during the read, and you  ↩
   can
break the program at any time using CTRL-C.
```

1.18 Grem2File

```
The usage is a bit like rob2file:

grem2file <disk unit> <filename> [starttrack] [endtrack]

For the moment, this utility works with all the Gremlins games I've tried,
and I think it will work on every Gremlins game that is not DOS copiable.

- Lotus Turbo Challenge
- Harlequin
- Zool ECS AGA
- Zool 2 AGA
- Supercars 2
- Switchblade 2

So you see, it was worth writing it! :-)

I recently updated this tool so reads are system-friendly and there won't be any
DSKRDY problems. Moreover, the pointer will not freeze during the read, and you  ↩
   can
break the program at any time using CTRL-C.
```

1.19 PsyDir

```
                Allows to read a psygnosis directory to see if the format is  ↩
                   rippable by
                rippsyfiles
                .
The format is simple:

$0000: Reserved                    $000C: $4000
$0010: FILE1 (null terminated)     $001C: $321A (file length)
$0020: FILE2 (null terminated)     $002C: $37FA (file length)
...
$00x0: $FFFFFFFF marks the end of the directory symbols.

$0C00: successions of bytes. 00 marks a Reserved block (not used), 01 marks a  ↩
   FILE1 block,
and so on. FF marks a spare block.
Blocks are $400 in size. Be careful, the blocks are not sorted from lower to  ↩
   higher.
There are files that come in several parts. I had trouble with SOTB3 disk 1 on  ↩
   this point.
Anyway, my ripper does the hard work.
When doing a HD-install, the file read patch must be inserted in a place where the ↩
    code looks
like that:

        subq        #1,D1        <- insert code here, at first subq
        bmi        label1
        subq        #1,D1
        bmi        label2
        subq        #1,D1
        bmi        label3
        subq        #1,D1
        bmi        label4


Usage:

psydir <diskunit> <filename>

will create an hex file containing the directory of the disk.
```

1.20 The Cadaver case

```
I've got the original disks of Cadaver and the Payoff data disk, but I noticed  ↩
   that I could
not make it work with my accelerated amiga. So I decided to patch it and I finally ↩
    succeeded,
even if I still don't know what made the game crash exactly (the stack value was  ↩
   not right).
Some users tested the patch and the hd loader (with patch) and were happy with it  ↩
   (Karadoc
```

```
appeared and they could move and play with the objects) but the saves and loads ↩
    crashed their
amigas. In the beginning, I thought it was an insulate problem, but at least five ↩
    or six users
reported me the same bug. I tested it but I must admit that it always worked OK ↩
    for me, both
with my 68060 card and with the vanilla 68EC020. One told me it was a floppy drive ↩
     problem,
and another said that there were a bug in the disk loading routine for loads and ↩
    saves.
They're both right I think. The load/save routine is too much hardware dependent, ↩
    which means
that there must be temporisation loops which fail if the CPU is too powerful, and ↩
    the drive
too weak.
The other version of Cadaver (released 1992) suppressed this problem.

I finally decided to patch the game so it's now possible to save games to HD.
I heard that it corrected the floppy save crash with the first version.
```

1.21 Quitting games

```
I recently succeeded in making some games quit and return to the OS, restoring the ↩
     display,
VBR, caches, interrupts, and freeing all the memory.

The F10 key is often used to quit games. It sometimes do not work during the ↩
    introduction,
or loading, because the keyboard interrupt handler is not installed yet, but it ↩
    works
during the game. Most of the time, I did not use the ESC key because lots of ↩
    software use
it to give up the current game, but sometimes F10 is used for something else, and ↩
    so I use
ESC or F5 sometimes.

To return to the OS, I had to save a great amount of chip memory, so this option ↩
    needs memory.
If you don't have enough memory, this option will be disabled, or may crash when ↩
    you quit.

If you manage (as I do) to fully control game memory management (very rudimentary, ↩
     most of the
time an allocation of 512K of extension memory), the return to the OS will be ↩
    perfectly clean
(no alerts or GURUs), as my code restores all the chip memory that could have been ↩
     used by the
game.
The minimum is $80000 (512K) and the maximum is $200000 (2MB for AGA amigas).
That's a simple idea but it works great!

Be careful when re-sourcing the game startup that you remove or emulate all ↩
    resource allocation
related system calls during the game (only at the beginning).
```

```
Most of all remove the AllocMem and AllocAbs calls. Be reassured, the game will ↩
    give up using
the system as soon as it will know how much memory it can use and where.


After having quitted one of my loaders, I generally keep on working, assembling, ↩
    running
very demanding programs without any crashes (including running another game)
(actually original HD-installable games crash lot more often the OS after quitting ↩
     than
patched ones :-) only because they use system routines clumsily).
```

1.22 Hardware used

```
I use the following hardware:

- A1200/Blizzard060/16Mb/SCSI2/IDE/4Gb/ZipDrive/CD-ROM for developping and
  testing my installers.
- A1200/020/NoFast/IDE/4Gb for testing my installers (actually, it's the
  same A1200 but I press '2' at bootup :-) )
- A500/2Mb+external floppy+Action Replay MKIII for looking inside the ECS games.
- Cyclone hardware disk copier.
- Technics CD player to fill the long programming silences...

For some games, I didn't have to use the Action Replay MKIII, or I simply couldn't ↩
    .
I managed with them because they were simple, but sometimes patching is impossible
without such a cartridge.
Now I use the excellent HRTMon 2.0 registered by Alain Malek. A demo version (very ↩
     powerful
already) is available on Aminet. It even works on my 68060.
The other monitors I tried just crashed (BigBrotherMon, ThrillKill, Action Replay  ↩
    V).
```

1.23 Software used

```
Here is a list of the software I use to create my programs.
Some are seldom used, but are necessary for some games.

- Pxhass for the assembler
- AsmOne for sensitive routine testing (disk data decoding)
- IRA for the executable and binary file re-sourcer
- HRTMon registered for hacking on the A1200/060 (much better than the
  Action Replay software, even on a 68040/68060). It really helps me a lot.
- XFD libraries for the unpacker(s)
- GNU make for the project builder (I recommend this one)
- GoldED registered for the editor
- HexED for the Hex Editor (by the author of MAME !!)
- RO registered for file manipulation
- XCopy 6 and Cyclone registered for diskcopies
- Action Replay V software version for some hacking on the A1200 (VERY limited)
```

```
I think that UAE (the very good Amiga emulator, by Brendt Schmidt, freeware,
now ported on almost all computers, even the amiga) would be a powerful and
cheap tool to debug and hd install games (even better than the Action Replay
MKIII cartridge).
But it's lacking tools for the moment (breakpoints, hacking functions, speed :-) )

That's just an idea to justify the existence of UAE on the Amiga...
Unfortunately, I experimented problems with UAE Amiga from S. Devulder, but it's ↩
   great
portage work.
For the moment, I'll use my hardware and HRTMon...
```

1.24 Documentation

```
- ROM Kernal autodocs, devices, libraries and hardware manuals
- The Amiga Bible
- Some hacked and re-sourced programs
- AGA.Guide (from Pandora, found on aminet). Excellent and FREE.
- Various 680x0 docs found on aminet
```

1.25 Patching

```
                HOW TO MAKE OLDIES WORK

Don't panic. If a game refuses to work, there are ways to manage with it without
necessarily writing a specific patch.
You can use degrader programs.
There are many of them on aminet. Most of the time they're freeware or shareware.
I can quote: KillAGA and WBKillAGA, TUDE (I recommend this one), SetChipRev,
Degrader, RunOldies, RunLame, Embedder, BootUte and I forget some for sure...

Degraders alter:

- Caches (not all of them for the 68060)
- VBR (set to 0 to avoid problems)
- Fastmem. They can block fastmem so you can run CD32 games without problems ( ↩
   Oscar).
- type of mems. They can fool the OS to make the game believe in fake fastmem or  ↩
   chipmem.
- Display (OCS, ECS, AGA)
- Sprites (ECS, 70ns, 35ns)
- MOVESR exception in user mode (trapped and replaced by MOVECCR). Seldom useful,  ↩
   because
  games often run in supervisor mode, but useful for OS compliant old apps.
- Some return values of Kicks higher than 1.3 to make them more 1.3 compatible.

For instance, TUDE makes all this above possible

But they cannot fool the computer on some low-level points
(roughly sorted by frequency of appearance in games):
```

```
                 Dumb blitter coding

                 CPU is too fast

                 Stackframe structure

                 Fastmem at $C00000

                 1MB chip as a maximum

                 24 bit addresses

                 Prefetch bug

                 Unhandled interrupts

                 Freezes (on keypresses)

                 Weird ROM accesses
               And they don't support 68060 specific problems:


                 MOVEP is emulated

                 CacheControl is ineffective
               Maybe I forgot some, but that's already a lot :)
By using the system, one could have avoided 95% of the problems...


                 Find the source of a bug
```

1.26 It's up to you Mulder

```
               1. The game works but the objects on screen flash/leave dirty ↩
                  pixels on screen,
and the game possibly crashes/freezes with gfx problems after a while:
99% sure it's a
               blitter
                problem (can be unnoticeable on 68060 while it appears
on a 68020 or 68030 because (I think) MMU disables caches in chipmem on the 060)

2. Game is too fast, music does not play properly, disk routines fail to work:
it' s a
               speed
                problem.

3. The game crashes neat (without gfx bugs). It can be a
               stackframe
                problem
(Cadaver, SWIV, Laser Squad)

4. The game crashes neat (without gfx bugs).
It can also be a old fastmem problem.
```

```
Hint: If the game mentions "1MB of RAM" and you don't see any system memory stuff ↩
   at bootup,
then there are 90% of chances that the game pokes $C00000 to see if the zone is ↩
   valid.

5. If the game works OK on an accelerated ECS amiga, and crashes on a AGA amiga, ↩
   then it's
very likely that it's a
                1MB chip assumed
                 problem (Lotus II, Supercars II).

6. If the game works fine on a 24 bit amiga (like a A1200/68EC020) and has bugs on ↩
    non-EC
cpus (strange behaviour, crashes during weapon changes...) then there's a lot of ↩
   chances that
it's a
                24 bit addresses
                 problem (Z-Out, Xenon2, Ruff'n'Tumble, Agony...).

7. It can also be a
                prefetch bug
                . 24/32 bit has no importance in this
case. Only the prefetch size of the CPU is to be considered (e.g the problem will ↩
   occur
on a 68030 as well as on a 680EC30).

8. If the game does not use the system, but crashes on Kick 3.1 while it works ↩
   fine with
other kicks, then it's a
               Weird ROM accesses
                problem (Gods).
(the kick versions are just an example. A game can work with Kick2.0 and 3.1 and ↩
   fail with 3.0)

So you've got to test on many configurations to see what's the problem.
As you see, AGA chipset is not the main problem. I laugh when I hear "AGA-fIxEd". ↩
   It's
totally wrong to say that in most cases, because most of the time the game did not ↩
    work
with ECS upgraded configurations. I would prefer 680x0 fixed.
```

1.27 HD Installing/Loading techniques

```
                Writing installation program

                Using disk file makers

                Floppies holding files

                Starting the game

                Loading virtual tracks
```

```
Loading virtual files

Changing disks

Modifiying memory detection

Handling supervisor mode

Removing protections

Switching the drive led(s) off

Removing DSKRDY code

Crunched code
```

1.28 Dumb Blitter Coding

```
This is a very frequent problem on the amiga. You try a game, and
the character flashes, or the game crashes after 2 seconds, screen
is garbaged.
In most cases, the game can be played, but the graphic bugs are
annoying. One could think the AGA chipset is the cause but it's not.
For proof test the faulty game on an accelerated A2000 and the result
is the same.

The problem is that programmers often don't wait for the old blit to
be finished before starting another. This is no concern on A500/68000
because they know the blits are over due to the terrible speed of the
68000, but when you upgrade to A1200/020 you see the good coders.

I coded lots of installs on 68060, and I thought there would be no blitter
problems for 68020-030-040 processors if there weren't with the 68060, and
that's where I was wrong.
On my 68060 Blizzard card, the chipmem access is terribly slow compared to
even a standard A1200/020 with caches on or a Blizzard 68030-IV card.
Some blitter errors can be unnoticeable, then (Warzone).

There's also some Bltpri configuration which seems to change behaviour either
setpatch is run or not...

In most cases, it's better to fix those problems.


WHAT TO SEARCH:

You've got to search write accesses to $DFF058 blitter register.
It can be found in various flavours in the code:

    move.w   D1,$DFF058

    or

    lea      $DFF000,A5
```

```
    ...
    move.w   #$56,($58,A5)

    or even

    lea      $DFF048,A6
    ...
    move.w   #$56,($10,A6)
```

Sometimes it can be a pain in the ass to find those instructions, as you can see.

HOW TO FIX:

You've got to make the CPU wait before or after the blit is performed.
It sounds natural that if you wait after, you'll lose CPU cycles since non-blitter related stuff has to wait. That can slow down the game a lot (Lotus 3)

If you wait before the blit, non-blitter code will be able to execute in parallel with the DMA blit, and the CPU will wait only in the case the old blit is not over when you reach the new blit instruction (necessary synchronization between CPU and Blitter).

To wait for blitter operation to complete:

```
wait:
      btst   #6,$DFF002               ; DMACONR
      beq.b  wait                ; wait until blitter DMA is over
      <make the blit>
```

Which is (almost) equivalent to the WaitBlit() routine of the JST package.

1.29 CPU prefetch problem

All processors of the 68K family have got a prefetch feature. The processor assumes that the code will be executed without break of sequence, so it prefetches the instructions to avoid memory accesses, and this feature cannot be disabled, unlike the cache feature that can be controlled.
So disabling the caches will not solve all the self-modifying code problems...

On a 68000, this code will work properly:

```
        move.w        #1,moveinst+2
        nop
moveinst:
        move.b        #0,D0
        ...
```

I mean that the value in D0 certainly be 1 (self-modifying code)

```
On a 68020 this still seems to work, at least the first time, and all the time
if you disable the instruction cache.

On a 68060, this code will not work (D0 will be still equal to 0!!) whether the ↩
   caches
are on or off. This can be harmless but can also lead to strange behaviour
if the instruction dynamically modified is a branch (I saw it in Elf!!)

In this case there's no other way than patching the code 'by hand', by breaking
the instruction flow (e.g by a TRAP or a BSR.B). Harry first noticed it in Vroom.
I used this technique efficiently in Elf.

Hint: coders insert NOPs in the code like the example above to be sure prefetch
will be KO on 68000. Search for NOPs in the code, and you'll be surprised to find
interesting things (CPU dependent loops, prefetch, cracked software :) )
```

1.30 24bit addresses

```
In non-EC CPUs such as 68030, 68040, 68060 (not 68000 nor 680EC20), an address is ↩
   coded on
32 bits, whereas 68000 and 680EC20 only take the 24 lower bits in consideration ↩
   when accessing
to memory, for data or instruction fetch.

e.g: on a 68000, if you want to jump to $4000, you can code:

JMP $4000 (the simplest)
or
JMP $xx004000, where xx can be anything different from zero (the stupidest)

Then, JMP $FF004000 will jump to 4000 (the PC will be equal to $FF004000 but the ↩
   instuctions
will be fetched from $4000 and so on and it won't crash.

Conversely, as real 32 bits CPUs don't mask the most significant byte, the same ↩
   instruction
JMP $FF004000 executed on a 68040 will cause the cpu to fetch the instructions ↩
   from
$FF004000, which will probably cause a superb crash.

But the programmers never code JMP $FF004000 instead of JMP $4000, I hear you say.
Yes, but they often use address tables (for fast switch/case) like this:

        move.l        D0,D1
        lsl.l        #2,D1
        lea        addresstable,A0
        move.l        (A0,D1.L),A1
        jsr        (A1)                <--- jump to fetched address
        move.w        #$95,D1
        moveq.l        #0,D0

addresstable:
        dc.l        $00004000,$00004046,$0000502A,...
```

```
A very convenient technique, except if the programmer has the stupid idea to use ↩
   the
unused most significant byte of an address (for instance at location (addresstable ↩
   ))
to store 1 byte data, such as a counter.
Then, the JSR will only be correct if the value is 0, else it will crash.

To patch this kind of error, the harder is to find it. It was the cause of the ↩
   crash
of Xenon2 and Z-Out, for instance, but only when a special bonus was taken.

On Agony it was harder to detect: some ennemies cannot be killed on a 68060, even
with all caches off. Still a 24 bit problem.


Once found, you can modify the code as follows:

        move.l        D0,D1
        lsl.l        #2,D1
        lea        addresstable,A0
        move.l        (A0,D1.L),A1
        jsr        My24BitPatch
        NOP
        moveq.l        #0,D0

Add NOPs as you won't have the room for the jsr (6 bytes vs 2 for JSR (A1)) and
copy some original code you overwrote:

My24BitPatch:
        move.l        D0,-(a7)        ; save D0
        move.l        A1,-(a7)        ; save A1
        move.l        A1,D0
        and.l        #$00FFFFFF,D0        ; filter the MSB (only with data ↩
           registers)
        move.l        D0,A1
        jsr        (A1)                ; jump
        move.l        (a7)+,A1        ; in case the game uses MSB of A1 (suckers ↩
           !!)
        move.l        (a7)+,D0        ; restore D0
        move.w        #$95,D0                ; original game code
        rts                        ; return

This example was adapted from Xenon 2 patch (the proof it works!)
```

1.31 Fastmem at C00000

```
Some games assume that you've got fastmem, and that it's located at $C00000.
This worked fine for A500/A2000 with cheap "fast"mem expansion, but it's no longer
true.
The best solution would be a degrader using the MMU to remap those addresses to a
block of configured fastmem. It would be a very good and convenient solution but
it does not exist for the moment. Besides, the gigabytes of (compressed) ↩
   documentation
about MMU programming that we can access are not enough for somebody to give
```

```
that feature an attempt (one of the most complicated english sentences I've ever ↩
    wrote).

While we're waiting for the MMU messiah, we can try to find where the game writes ↩
    to
those addresses (possible with an Action Replay MKIII), change the base pointer
and that will work.
Some others errors come from programs which don't want to assume the $C00000 ↩
    fastmem
blindly, but the tests of which are crap, and the game believes that there is
fastmem at $C00000 (Supercars 2 behaves like this). From that point, it's the same
problem than before, except that it's easier to patch the memory detection routine
and insert a prioritary check for chipmem at $80000. If found, this chipmem will
replace $C00000 ghost ram.

This example comes from SuperCars2. The patch is not included cos' it does not ↩
    apply
to the original game. Sorry. If you've got the Supercars 2 original copy, please
send it.

A test under UAE is the best way to figure out if the game needs this $C00000 ↩
    fastmem.
```

1.32 1 Megabyte of chipmem

```
A problem encountered twice is a faulty chip memory detection because of the ↩
    unexpected
fact that AGA amigas have got 2MB chipmem.

Some games try to find fastmem at $200000 (that exists on some boards).

        move.l #$AAAAAAAA,$200000      poke in $200000
        cmp.l  #$AAAAAAAA,$200000      re-read to check if the address is valid
        bne    nofastat200000
        ....

This detection works provided the chipmem size does not exceed 1Mb ($0 to $FFFFF).
With a AGA amiga like a A1200, chipmem can be found from $0 to $1FFFFF.

When you poke in $200000 on a A1200 without fastmem at $200000, there is a mirror
effect and the write address is decoded as $0. So, the re-read is ok, and the
game trusts at least 512K of memory from $200000 to $27FFFF. It actually stores
the data from $0 to $7FFFF and it crashes very efficiently, as program code and
stack are in that zone 99% of the time.

The 'safe' thing to do would have been:

        move.l #0,$0
        move.l #$AAAAAAAA,$200000      poke in $200000
        cmp.l  #$AAAAAAAA,$200000      re-read to check if the address is valid
        bne    nofastat200000
        cmp.l  $0,$200000              test the mirror effect
        beq    nofastat200000

The mirror or modulo effect is detected.
```

```
This error was found and corrected in Lotus Turbo Challenge 2 and 3, from Gremlins ↩
    .
```

1.33 CPU is faster than a 68000/7MHz

```
That's a fact, and that's why you upgraded, actually.

But some games don't appreciate this at all, because the timing loops
were calibrated for a A500/68000, and are not sufficient for a faster CPU.

Disabling the caches can be the solution, and you can also disable fastmem, to be
sure that the program won't run faster due to real fast memory.
You can also increase loop timing, or make a real timing for instance with
VBLANK interrupts (it's not easy) or beam counter (easier)

e.g: First Samurai does not work even with a 680EC20, and I'm 99% sure speed
is the problem. Mr Larmer patched it god blesses him :)

e.g 2: Jaguar XJ-220 believes the race is over after 1 second playing on a 68060.  ↩
    It was
already too easy on a 68030 because the speed of your car was calculated in a  ↩
    different way
than the computer cars (CPU speed vs CIA or VBLANK timers). I heard that the same  ↩
    problem
occurs in Hard Driving.

e.g 3: The music does not play properly in some games even with caches off. That's ↩
     because
a very widespread code uses a stupid delay loop:

loop:
        dbf        D0,loop

Without NOPs, 68020+ optimizations make the loop very short, and the music fucks.

A lot of reverse engineering is compulsory in those cases (sometimes made harder  ↩
    by protections)

Good luck.
```

1.34 STACK FRAME

```
Stackframes are different between processors of the 680x0 family.
I'm not a specialist, but don't make assumptions about the stack frame
except for Push Address + RTS, which works in all cases.

When I patched Laser Squad I discovered that the game pushed SR on the
stack (2 bytes), pushed return address on the stack (4 bytes), then
jumped to a location which called RTE to return. It works on a 68000,
but crashes on any other processor I know (at least in this case)
To correct this, the better way is to make stackframe inependent from the
type of processor by using a TRAP (the JSR equivalent of RTE).
```

And it worked ! :-) Much cleaner.

1.35 UNHANDLED INTERRUPTS

Some programs don't care about handling interrupts which could happen
according to the way they set INTENA and SR, but never do on a 68000
due to low CPU speed.
That's the case for high-level interrupts, like level 6 (vector $78).
If the processor is too fast, problems can occur.

E.g in Pinball Dreams, a level 6 interrupt is triggered by CIA-B with a 68060,
but it does not happen on a 68000 or even on a 68020. The level 6 vector
was not initialized and left to a value in ROM -> It freezes.

The solution was to patch the level 6 vector to a routine we coded ending
by a RTE, but RTE alone is not enough, because we've got to acknowledge
the interrupt, else it will occur relentlessly.
For CIA interrupts, just read the ICR and the interrupts are acknowledged
(see the source for the patch of Pinball Dreams).
For VBLANK or custom chip interrupts, acknowledge by writing in the INTREQ
register ($DFF09C). Refer to the custom chip documentation (e.g: AGA.guide, on
aminet) for all the details.

Unhandled interrupts are quite difficult to detect. Check that all interrupts ↩
vectors
are assigned to a non ROM value.
JST detects them automatically with the DEADLY option, and if DEADLY is off, it ↩
will
acknowledge them, so you don't have to care about them :)

1.36 Freezes (on keypresses)

Some games work fine until you press a key. Then it freezes and you're forced to ↩
reboot.
But the music is still playing, and some animation can continue.

That sounds strange and that is, actually.

Keyboard interrupts have got a priority level of 2, while VBL interrupts
(often used by tracker routines) have got a level of 3. Which explains that
the music can continue playing.
The keyboard interrupt was not acknowledged, and it happens all the time. The
program can't continue. Only higher interrupts can run.

This is often caused by an acknowledge too soon before the RTE:

E.g:

```
KbInt:
        move.w        #8,$DFF09C        ; acknowledge interrupt
        ...
```

```
        ...
        RTE
```

Moving the acknowledge instruction just before the RTE can be enough.
If this does not work, try replacing #8 by #$7FFF (that's a bit strong...)

Noticed (and fixed) in Ninja Spirit and R-Type2

Interrupt 3 can behave the same too. Same solution (try #$70 before trying #$7FFF)

Noticed (and fixed) in Z-Out

1.37 WEIRD ROM ACCESSES

Some games rely on some values in ROM to work properly. E.g I did not
understand why the great game 'Gods' worked OK from OS1.3 to OS3.0, but failed
on OS3.1, on the same computer (worked OK with softkicked OS3.0, failed on
on-chip OS3.1 !!).
Harry gave me the solution recently: the game reads in $FFxxxx (i don't
remember now) and I really don't know why. By luck, it worked until OS3.0
but the game seems not to like the values returned by OS3.1. This is now
fixed in my loader (and soon in Bert's one)

I don't even talk about games calling directly ROM addresses (Gravity
Force) or poking in non documented exec strucures, copperlists...

A possible explanation of those accesses in non-DOS games is a
protection against hardware freezers like Action Replay or Nordic Power.

There was a check in $F00000 for the value $1111 in the game
Pinball Dreams, maybe to detect such a device but:

1. It did not detect Action Replay III (hopefuly :-))
2. It detected my Blizzard 1260 and crashed willignly! (how stupid)

I saw some other accesses (including writes) in other games.
If you find routines looking like this, remove them, or it may
cause problems to other users of your patches.

If someone has got information about this, I'd be very grateful to
learn about it.

For the moment, the solution is 1.look, 2.get good value (using a 1.3 kick, or/and
another amiga), 3.remove, 4.imitate.

1.38 MOVEP

In the 68060 model, Motorola had the great idea to remove the MOVEP instruction.
It was a bit useless but some programs used it, and now they fail to work.
A software emulation exists but is not installed at startup and can be removed
if the program pokes into the interrupt vectors.

You'll have to find the MOVEP manually and replace them by 2 judicious MOVE.B.

SpeedBall2 has got this problem (not only this one, though :-( )

Sometimes my loaders crash with "LINEF/MOVEP (68060) encountered".

That does not mean that you've got a 68060 :-)

That only means that a LINEF exception occured if you've got a 680x0 (x<>6),
and a LINEF or a MOVEP occured if you've got a 68060, but I still have
to check how a 060 computer behaves exactly in front of a MOVEP.

1.39 CacheControl problems

CacheControl is not able to remove branch and writebuffer bits in the cacr register.
They often cause problem, even if I don't know exactly why (removing all the caches
but those sometimes make games crash). pcr register (new on 68060) holds a bit
controlling superscalar mode.

As said before, my installer own specific routines to remove those caches:

```
; *** Disable some '060 caches
; *** Left intact by CacheControl

; internal, only useful for 68060 cpus

DisCacheSup:
        MACHINE        68060
        ori.w         #$700,sr
        movec         cacr,D0
        andi.l         #$20800000,D0
        move.l         D0,old060cacr

        movec         cacr,D0
        move.l         #$20800000,D1
        not.l         D1
        and.l         D1,D0
        movec         D0,cacr

        movec         pcr,D0
        move.l         D0,old060pcr
        andi.l         #$04300100,D0
        movec         D0,pcr

        CPUSHA         BC
        MACHINE        68000
        rte

; *** Enable some '060 caches
; *** Left intact by CacheControl

EnaCacheSup:
        MACHINE        68060
```

```
        ori.w        #$700,sr
        movec        cacr,D0
        or.l        old060cacr,D0
        movec        D0,cacr

        movec        pcr,D0
        or.l        old060pcr,D0
        movec        D0,pcr

        CPUSHA        BC
        MACHINE        68000
        rte
```

In user mode, you must call those 2 routines with Supervisor(),
else you'll get a privilege violation. In supervisor mode, a trap will be ok.

If you've got some improvements to bring to those routines (they're totally ↩
   experimental),

                Contact me
               . I'd be glad to know a 68060 experimented
programmer.

If you dislike programming just make a script with 'cpu060 ns nw nb' to remove the
annoying caches.
I tend to avoid scripts, personnaly.

My HDInstall object file allows to control efficiently the caches, especially ↩
   68060 ones.

1.40 The floppies holding files

               Sometimes you can't start a game from HD even if the disk seems to ↩
                   be a normal DOS disk.
Don't begin to program too heavily if only a script with 2 or 3 assigns and ↩
   degrader
calls (TUDE...) before the executable would be enough.

Example: Banshee runs fine from HD:

-- cut here --

```
;cpu check 68040 >nil:
;if not warn
;cpu nodatacache >nil:
;endif
; for 68040 users

;cpu060 nd ns nb nw
; for 68060 users

Assign bans1: ""
Assign Banshee1: ""
```

```
Assign Banshee2: ""
Assign Banshee3: ""
Assign Banshee4: ""
picture.exe
bans.exe

-- cut here --

Of course you've got to let Disk1 in drive DF0: for the protection check.


Some game executable hold one or many "DF0:file1", "DF0:file2", etc, hardcoded ( ↩
    example:
F18 Interceptor, Exile AGA, PowerDrome).
To run them, take a hex editor, and replace all DF0: occurences by any 3 letter  ↩
    assign (F18:,
for instance)
After that, to play, type:

assign F18: "" ; or the assign you've chosen
"F-18 Interceptor" ; executable name

F18 does not work properly on 68060 unless you remove fast memory

This method can fail if the game uses the OS only to load the first files, then
uses its own routines. See
                here
                 if it this happens, but you can
choose
                this
                 solution (lower level, but no worry with files).
```

1.41 Writing installation program/script

```
                This part can be the easiest, as it can be one of the hardest!

When you decide to hd-install a game, there are 3 possibilities:

1.The floppies hold files, and a custom routine is used to read them
  (that's why it's not installable on HD and you cannot choose
                this
                 solution.
  But for the installation, it's the same. You can create an installer or an iconx
  script to copy the files in a directory where your loader will be located too.
  You can also consider that the disk is NONDOS and attack at a lower level (then  ↩
     see case 2)

2.The floppies are copiable with any copier (except maybe a special track).
  You can use any disk image creator (
                mine
                 is crap, but works) to
  create files holding the disk contents. Use a disk image creator in a script ( ↩
     iconx
  or installer). You can now install multi-floppies trackloading games.
```

```
3. The floppies are copiable in nibble mode only, or not at all (long tracks) and they
   belong to the games on the list (or same publisher, or design team):

   TEAM 17 (Rob Northern):

   - Project-X Special Edition
   - Superfrog
   - Body Blows
   - Body Blows Galactic
   - Body Blows Galactic AGA
   - F17 Challenge
   - Alien Breed II AGA
   - Arcade Pool
   - And probably many others (but not Assassin, Qwak, ATR, Alien Breed, which are
       disk2file-able)

   PROBE (Rob Northern, diskkey=0)

   - Mortal Kombat I
   - Mortal Kombat II
   - Primal Rage

   PSYGNOSIS (Basic directory structure)

   - Leander
   - Shadow Of The Beast 3
   - All lemmings editions
   - Probably some other games from PSYGNOSIS (but not Awesome, nor Agony)

   GREMLINS (Nibblecopiable)

   - Lotus series
   - Zool series
   - SuperCars II
   - Harlequin
   - Switchblade II
   - and probably all the Gremlins nibblecopiable disks

   You can use one of my
                special disk rippers
                to handle those files.
   If the rip succeeds for a game I did not write an install for, please e-mail me
       to
   warn me about it, and I'll be able to write an installer and a loader very
      quickly.


4.The floppies are copiable in nibble mode only, or not at all, and the different
    disk file
  makers fail on the disks.
  This is where the installation is hard to code. You've got 2 solutions:

  a) Rip the original game loading routine, understand it, and include it
      completely in
     your installer program.
```

```
  I made it for Backlash, Elite, Skidmarks, Lure of the Temptress... This method ↩
     is not
  really satisfactory because it uses hardware-coded routines, and can fail on ↩
     some
  machines (because of the caches, or DSKRDY not present on AT A1200s). Moreover, ↩
     the
  code has to be deeply understood and modified to avoid crashes or enforcer hits, ↩
      and
  you'll have to add some features (drive selection, check disk in drive...). I ↩
     gave up
  this method.

  b) read the tracks using the trackdisk.device in raw mode (see the rippsyfiles
  source code for an example) and then decode them with an assembler program ↩
     inspired from
  the original.

  In this second case, the harder part is to find the synchro and the bitshift
  on the rawread track, because system routines are only reliable with INDEX ↩
     synchro and
  not WORD synchro. The best thing to do in that case is to use the game loader to ↩
      get
  a synced track and then use Asm-One to adapt the decoding routine on the ripped ↩
     rawtrack
  (if you rip one which sync is shifted (read with trackdisk.device INDEX mode), ↩
     the syncs
  and raw data will be shifted and you'll have difficulties to view the data.
  Once your decode routine is ready, include it in your ripper and rip the whole ↩
     disk.

  This method b) has many advantages:

  - Only the decoding routine has to be understood
  - It does not use the hardware registers at all -> better compatibility with all ↩
      amigas.
  - trackdisk.device is easy to program, you can perform disk checks and drive ↩
     selection.
  - You won't have to freeze the system while you read the data
  - Errors will be handled better

  I used this method to rip R-Type 2 disk, and then I adapted it to recode ↩
     rippsyfiles,
  grem2file, and I will
  Both of those methods take time, but it allows to safely backup your non- ↩
     copiable disks
  on HD (no more fear to lose the data).

Look at the various examples provided in the package, and you'll see all the ↩
   possibilities.
```

1.42 Starting the game

```
- If the game loads from a NONDOS disk, the only thing to do is re-source the bootblock
  (the $400 first bytes of the disk) and include it in your code, emulating calls to
  DoIO() (-$1C8(A6)) by a routine reading from the diskfiles you've made.
  Sometimes the bootblock is not necessary, because it just reads a track and jumps to it.
  So you just have to imitate its behaviour (don't forget to flush the caches before
  jumping to a freshly loaded code)

My package includes a trackdisk.device emulation which only allows to read tracks, but
it's enough in all cases.

- If the game loads a DOS executable program, and then starts his custom floppy- load code
  (it's the case for SWOS or Cannon Fodder 2), you've got to re-source this loader and
  write one yourself. The loaders are generally small. Use ADis from Martin Apel or IRA
  from Tim Ruehsen to re-source them. Be careful: re-sourced code can fail to work.
  but anyway, it helps to understand how it works. IRA gives very good results. I could
  reassemble lots of big asm files created by IRA and it worked fine. I used Phxass with
  the no optimize option (OPT 0), else data sections resourced as code could be ' optimized'
  and the data would be corrupt.

You can also create a file in RAM: which is a copy of the loader executable, but with
a patch you inserted at some offset to allow you to get the control after the loader
has done its stuff. Execute this program instead of the original loader, and the trick
is done. This last technique is not the state of the art, you'll understand why, but
you're forced to use it sometimes (I began Cannon Fodder 2 HD loader like this)
Think about decrunching the executable, if you think it's crunched. Use XFD to do this.
It recognizes and decrunches lots of formats (Imploder 4, Crunchmania, PowerPacker, RNC,
ATN, ICE...)

Starting the game means you've got to set the caches/VBR/Display properly to avoid
the game behaves weird. My package include functions to open a PAL (default) or NTSC
screen, disable the caches you want, and set the VBR to zero (optionnal).

Hint from Harry: Some games rely on special values set in hardwareregisters at bootup,
eg Battleships needs Bit $e in UWORD $dff09a (enable general interrupt), Gauntlet 3
uses sprites, but never switches the spritedma on. There may be other examples.
```

You've got to examine the bootblock very attentively and try to reproduce exactly
the coldboot configuration.

1.43 Loading virtual tracks

Now, you've started the game. Great.
Now you've got to locate the disk loading routines entrypoints and parameters
(often registers, but it can be memory variables, or mixed). The authors of the
game protect their games, but they don't try to hide the loading routines too much ↩
,
because it has no interest for protection (except for special encoded tracks).

- If the game disks are not copiable or nibble copiable and you succeeded to ↩
create
reliable disk images , that means you know where are the disk routines and how
they work (you ripped them, don't you remember).
Insert your emulation and start the game.

- If the disk is standard trackload non DOS disk, you've installed the diskfiles
with a disk image maker, and you have to find the routines which load the game, ↩
not
to rip them, as you know how to read a standard track in a system-friendly way, ↩
but
to understand how it works.

The most famous loading routine is the Rob Northern loading code. There are ↩
several versions
of this routine but mainly there are 2 versions:

* 11 track version : size 901120 (STANDARD_DISK_SIZE)
* 12 track version (long track, non copiable, even in nibble mode) : size 983040
(S12_DISK_SIZE)
if it begins at track 0 (data disks) or size 970752 (B12_DISK_SIZE) for boot ↩
disks
(start at track 2)

The 12 track routine needs to be ripped to create the disk images (installation ↩
part).

I saw it in Body Blows, Project-X, Mortal Kombat I & II (same routine for MKI and ↩
MKII)

The advantage of those loading routine is easy recognition of them in the code.

In assembly language, it begins like this:

```
        movem.l        D1-A6,-(A7) (sometimes D1-A5)
        link        A6,$FFDC    (sometimes $FFDE)
```

In binary, you can search them as:

```
        48 E7 7F FC 4E 56 FF DC
or        48 E7 7F FE 4E 56 FF DE
```

more generally

```
        48 E7 7F Fx 4E 56 FF Dx

Searching 4E 56 FF will often give satisfaction.

Once found, you've only to patch this routine by the one provided in my package,
excepted that you've got to choose the right disk (if there are 2 or more disks).
You've got to substract from offsets sometimes too (12-sectors boot disks have got
2 11 sectored track for bootup from the OS).
The best thing is to look in the examples given (Mortal Kombat, Desert Strike and
Qwak are provided with full source).

The arguments program pass to this routine are:

A0  : Destination memory buffer, chip or fast memory
A1  : Destination raw disk data buffer, chip memory. Forget this if you HD-Install ↩
    .
D0.W: Disk unit (0 to 3)
D1.W: Sector beginning offset (sectors are 512 ($200) in size. D1*$200=offset in  ↩
    bytes)
D2.W: Length in sectors (multiply by $200 to get the real byte size)
D3.W: Command ($8000 or $0: Read, $8001 or $1: Write you have to handle this case  ↩
    separately)
D4.L: Disk key. See
                Rob2File
                for more information.

This D4 parameter is useless when the disk tracks are DOS (11 sectored RN routine)

It returns 0 in D0 if everything was OK, and a negative return code if not (disk  ↩
    error,
no disk in drive).

The ReadRobSectors() routine in my hd-utility object code matches exactly
arguments A0,D0,D1,D2, so most of the time you can directly call this routine.

        JSRGEN        ReadRobSectorsFast

It always return 0 in D0 (no errors assumed, all virtual disks in drives).


If you don't find this loading routine, you'll have to do the job yourself.
Sometimes it's not really necessary as a mere memcopy from disk image to
memory is enough to load a part (old games, see Backlash) and we don't
bother with the details of the disk loading routine.
Experiment with all this, it's the only way to make it.

NOTE: Sometimes the same disk routine is copied several times on the game disk:
Routine 1 is used to load the introduction program, which holds routine 2,
which will load the main program, itself holding its routine 3, and so on.
You have to test the game heavily to be sure no floppy code remains unpatched.
The Desert Strike example shows how hard it is to patch some games.
That one meant hard work for me.

A very simple example is also provided. It's my first HD installer: Qwak.
It has only one disk loading routine, directly patchable from the diskfile
(not packed), and the only disk (so no diskchanges) is a normal 11 sectored
nondos copiable disk. Moreover, there is no on-disk copy protection.
```

1.44 Loading virtual files

```
The games accessing files through their own OFS reading program exist too. Editors ↩
    do
this because they want to protect their games, or they wanted to get rid of the OS ↩
    to
handle memory as they want and it's really fustrating because you know that  ↩
   developpers
had the game on their hard drive, and that the PC version will have exactly the  ↩
   same data
files, but will load from HD without problems (Cannon Fodder 2).

You've got to locate the place where the game reads a file. Tools are provided in  ↩
   my
package to load all the files in a directory (no subdirectories yet) to mem, and  ↩
   recall
them during game. It was really hard work for me (lots of bugs to remove) only to  ↩
   patch
Cannon Fodder 2. After that I used it many times and I think it's totally bugfree  ↩
   now.

The hard part: if the game has got a save-to-disk option, you must filter files  ↩
   that are
to be saved, and bypass your routine to really write on floppy
(I don't know how to in-game write or read to hard disk), and read those files  ↩
   from
floppy too. See the Cannon Fodder 2 installer source. I've done it here.
```

1.45 Patching disk changes

```
It's a lot easier to patch a one-disked game than a multi-floppy one.

In the last case, you have to figure out how does the game do to read data
from the different disks (a variable set to 0,1,2, etc...), a header like MKD1,  ↩
   MKD2,
a file named disk1, disk2...

For games supporting external floppy drives, it can even be easier, as you only  ↩
   have
to virtually insert disk 1 in DF0:, disk 2 in DF1:, disk 3 in DF2:...
But some 3 disked games support only 2 floppy drives. You cannot use this method  ↩
   in that
case.
For the Rob Northen loaders with 2 disks, just patch the loading location by
ReadRobSectors. Register D0 shows the selected drive, which will match the  ↩
   selected
disk.

Sometimes, you have to patch 2 routines: the loading routine and the routine which
tells you to insert disk number x in the drive.
```

```
Just get the number x wanted, put it in a current_disk variable and make the  ↩
   program
believe the disk has been inserted. Then, when the loading routine is called, set  ↩
   the
disk number you stored in current_disk and load from it.
It works very well.

From a certain point in the program, the first disk can become useless
(e.g. Chaos Engine, Cadaver). Then, one of your routines located at the right  ↩
   place
can set disk number to 2, and it will remain like this until the end.
Useful when you can't find disk number in the code, and the game reads only
from DF0:.
```

1.46 Configuring extension memory

```
                A game which need 1MB of memory (or more) will try to figure out  ↩
                   where memory locations
are, either using the system (AllocMem, memory lists) or poking in assumed  ↩
   location to see
if there's memory here (see
                the 2MB chip problem
                 for details on this
last technique).

For games using only 512K of chipmem (old games like Xenon2, Backlash), this  ↩
   problem does not
exist, but if you want to install a quit option on your 1MB games, you have to  ↩
   patch the memory
allocation, since their methods are ALWAYS dirty. Even the ones using AllocMem  ↩
   make a mask
with $FFF80000 to get rounded memory pointer, and also to be sure not to waste  ↩
   memory (the floppy
games are designed for 1MB A500 amigas). If you patch the memory tests, you can:

1. Force the game to use FastMem (sometimes games use location $80000 for 1MB chip ↩
    amigas)
2. Protect your memory. The game will not write or read outside the zone you fixed ↩
   ,
   which makes quit option more efficient.
3. Be sure that all those AllocMem are removed. When you quit, there's no  ↩
   allocated blocks left,
   and you don't lose memory.
4. I recommended to use AllocFakeMem() in V1.0 of this guide, but I removed this  ↩
   function because
   I discovered that it caused crashes. Instead of that you can choose an absolute ↩
       location in
   chip memory (in the zone you'll save with SAVEOS_DATA). AllocFakeMem was only  ↩
      useful in
   bootblocks so you should not miss this function too much.
5. Can even make the game useable: some memory checks can crash the machine, and  ↩
   some can
   allocate 32 bit fastmem that the game won't stand!!
```

```
For instance, a game will use the 512K chipmem (from 0 to $80000) and 512K of other mem
($80000-$100000, or $C00000-$C80000). This last zone is very rarely hardcoded in the program.
It is generally referenced by a pointer located in chip memory. Find the pointer, assign it to
the allocated value and you succeed.

A routine (AllocExtMem()) exists in my HD package to allocate a memory zone dedicated to
extension memory. The interest of this routine is that the quit procedure frees the
zone automatically.

See the Assassin source code for an example.

WARNING: Some games like Mortal Kombat II don't like 32 bit fastmem extensions at all.
You'll have to locate mem extensions in chip memory. That means 512K chipmem amigas won't
be able to run the game (or adapt the routine to be able to use 24 bit fastmem)
```

1.47 Handling supervisor mode

```
99 percent of the games use supervisor mode to execute. The reason is simple: they want to control
everything in the amiga. To do so, they use pieces of code like this:

        move.l        #supcode,$20.W        ; privilege violation trap
supcode:
        move.w        #$2700,sr        ; go in supervisor more. Trapped the first time, then executed
        ...

or:

        move.l        #supcode,$80.W        ; trap 0 address
        trap        #0
supcode:
        ...

or (system friendly):

        move.l        _SysBase,A6
        JSRLIB        SuperState        ; jsr        -$150(A6)
        move.l        D0,_userstack        ; optionnal


If the game switches in supervisor mode at a moment, you HAVE to do it before the game
does, with a macro called GO_SUPERVISOR. It will execute the system friendly code above.
If you don't do it, the quit option could fail.
```

Generally, games switch in supervisor mode early in the execution. Do it yourself and try to skip this part (as with 512K extension memory allocation).

1.48 Removing protections

There are several types of protections:

- Non copiable disks (all the sectors): those are "cracked" when you install them to disk
- Copiable disks (even DOS) with a faulty sector: physical copy needed for this sector, but it generaly holds no data. Sometimes hard to remove because of a special crypted routine.
- Copiable disks (even DOS) with passwords you have to enter to check you own the original.

Sorry, my goal is not to teach you how to crack games. Learn it by yourself. I don't like doing this anymore (although it's necessary).

1.49 Switching drive led off

Once you made the HD-Installer, you want everything to be perfect, and the drive led still lits sometimes. You can switch it off with a move.b #$7F,$BFD100. Be careful though, if some disk code remains, the program can wait forever for a disk to be ready even if this would have worked with no disk in the drive...

1.50 Removing DSKRDY code

If you switched the drive led off, it can happen that the game freezes, because this kind of loop is in the code:

```
loop$
        btst        #5,$BFE001
        bne        loop$
```

This checks if the disk is ready for ready. If you switched motor/led off, the disk will never be ready and it will loop forever. Moreover, models A1200 from AT don't have DSKRDY flag (hardware bug from Escom), and that's why lots of games fail to work from floppy on this machine.

```
Solution: remove this useless code.
```

1.51 Managing with crunched code

```
The most used cruncher in games is the Rob Northen Cruncher. It allows unpacking
of the data overlaying the crunched data. You only need one buffer and it's  ↩
   convenient,
but the crunch rate is poor. However, it can bother crackers or hd-patchers, as  ↩
   the code
is not directly patchable from the diskfiles.
The best solution is to put a patch before the cruncher exits to modify the code.

You can also use the RNCDecrunch() routine to decrunch Rob Northen Cruncher
files (only those). ATN! crunched files can be decrunched by ATNDecrunch()
(alternate Rob Norhten cruncher). Some RNC code is crunched (see Walker). My hdlib ↩
    includes
a function to decrunch encrypted files (provided you've got the key, hehe :-) ).
If you don't know the key, you can use 'HackProPack' (XFD package) to decruch the  ↩
   files.
I recently added a decruncher for Gremlins games (used in Zool series)

Example: the cruncher ends like this:

        movem.l         (A7)+,D0-A6
        rts

This takes 6 bytes long. Replace it by a JMP Patch ($4EF9 patch address)

Patch:
        ---- do your stuff here ----
        ---- flush the caches if necessary ----

        JSRGEN        FlushCachesHard ; flushes caches in a hardware way (no  ↩
           CacheClearU())
        movem.l         (A7)+,D0-A6        ; the original code.
        rts


Note: It's always better to decrunch the data in your code, since it's in fastmem  ↩
   most of the
      time, but the original runs in chipmem. So your loader increases decrunch  ↩
         performance.
```