AMP

COLLABORATORS

	TITLE : AMP	

ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		January 19, 2025	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

AMP

1.1 AmiDog's Movie Player - The Movie Player for your PPC Amiga!

```
AmiDog's Movie Player v1.23, 990921

      Copyright 1998-99

   Mathias "AmiDog" Roslund

 Introduction           - What is this?
 Requirements           - What does it require?
 Features               - What can it do?

 Installation           - How to install?
 Usage                  - How do I use it?
 Menus                  - What can I do?
 Speed                  - How fast is it?

 Shareware              - What does this mean?

 Disclaimer             - Who's responsible?

 Development            - How is it done?
 Bugs                   - Are there any known bugs?

 History                - What's new?
 Future                 - What will be added?
 Contact                - How to contact me?
 Thanks!                - Which have helped me?

Visit the homepage at http://www.amidog.com/amp/
```

1.2 AmiDog's Movie Player - Introduction

```
This is a Movie Player for Amigas equiped with a PPC board.

I started writing this in november 1998 since there were no MPEG players
for PPC which had good AGA support. And as you can see, the AGA support
```

is probably one of the best in any movie player for the Amiga.

Already from the beginning I planned to add support for more media types than MPEG, but it's not until recently (v1.20) that I've had the time to do so.

This product is released as Shareware, which means that you can freely use it for a period of 30 days, but then you must send the author a small fee. There are also some restrictions in the unregistered version, it only support MPEG.

Now start the player and enjoy the speed of your PPC board!

1.3 AmiDog's Movie Player - Requirements

Hardware:

* Amiga 1200/3000/4000 with a PowerPC board
* 16 MB RAM
* OS 3.0+

Software:

* WarpUP v4.0
* CGFX v3 or later for CGFX support

1.4 AmiDog's Movie Player - Features

It currently supports the following multimedia types:

* MPEG1/2 video with sound (optional) and automatic frameskip.
* FLI/FLC (most chunks supported, please report any problems).
* AVI (only a few codecs is supported currently).
* QT (only video currently).
* IFF ANIM (not yet, not started).

NOTE: The unregistered version only got MPEG support.

Supported encodings:

Video:

(AVI, QT)	Radius CinePak	(CVID)	8/16/24/32 bit
(AVI)	Microsoft Video 1	(MSVC)	8/16 bit
(AVI)	?	(YUY2)	16/24/32 bit
(AVI, QT)	JFIF JPEG	(JPEG)	8/24 bit
(QT)	Apple Video	(RPZA)	16 bit
(AVI)	Microsoft RLE8	(RLE8)	8 bit

Audio:

(AVI)	PCM, Mono & Stereo	8/16 bit

```
 NOTE: Even though you actually can play 8bit FLI/FLC/AVI files in any higher
       depth (15/16/24/32 bit), it's not recommended since it's slower than 8bit.


The features include:

 * Fast AGA support with 10 HAM modes.
 * CGFX support using direct videomem access.
 * Demuxing AsyncIOPPC routines (MPEG).
 * General AsyncIOPPC routines for FLI/FLC/AVI/QT etc.
 * Fast audio playback using audio.device.
 * Frameskip (automatic, MPEG only).
 * CLI support and GadTools GUI.
 * Window support (CGFX) with optional speed hack (FAST!).
```

1.5 AmiDog's Movie Player - Installation

```
Just copy it to any place on your HD.

The keyfile you get when you register should either be placed
in the same directory as the AMP executable or in the S: assign.
```

1.6 AmiDog's Movie Player - Usage

```
1. start shell.
2. stack 1000000 <ENTER>.
3. amp -gui, amp "filename" or amp -? (for help)<ENTER>.

NOTE: If AMP crashes, try to increase the stack!

Quit playing by pressing the left mouse button (LMB).
```

1.7 AmiDog's Movie Player - Menus

```
 File   General   Output   MPEG   Misc


Please click on a menu item.
```

1.8 AmiDog's Movie Player - Menus - File

```
 File   General   Output   MPEG   Misc


Open - Select a MPEG to play.
```

Play - Plays the selected MPEG.

Edit Prefs - Here you change the screenmode preferences.

Save Prefs - Saves the preferences.

About - Clears the listview and prints the about information in there.

Quit - Quits AMP.

NOTE: When the preferences are saved, not only the screenmode preferences are saved,
but also all current settings like HAM Quality, ColorMode etc.

1.9 AmiDog's Movie Player - Menus - File - Edit Prefs

Back

When you choose Edit Prefs, AMP will open a special window in which you can tell AMP which screenmodes to use.

This is the list AMP will use when you have selected From List as the screenmode to use, AMP will start at the top and move down until it finds a screenmode which is big enough to show the whole MPEG. One special case exist, adding a screenmode with width and height zero (0) will make AMP use this one in case it can't find one that fits.

The list is automatically sorted to make sure that AMP always selects the right screenmode for the loaded MPEG.

Below the list there are three buttons, Add, Modify and Delete which will do just that, add, modify or delete the selected item in the list.

Above the Save, Use and Cancel buttons are a set of edit gadgets, which you use to
modify or add screenmodes. By pressing the Choose button, you'll be able to select
a screenmode using an ASL requester.

NOTE: You can not have two screenmodes with the same width and height in the list !

1.10 AmiDog's Movie Player - Menus - General

File General Output MPEG Misc

Screenmode:

* From List - AMP select the best one from the screenmodes list.

```
 * Best Mode - Uses a OS-call to get the best screenmode.

 * Requester - Let's you choose the screenmode to use from an ASL requester.

 * PAL       - Uses PAL Lores/Hires depending on the MPEG size.

Limit FPS:

 * From Stream/5/10/15/20/25/30/Maximum - How fast the MPEG should be played.  ↩
    NOTE: This
                                          setting is ingored when audio is  ↩
                                             enabled since
                                          the audioroutine generates the correct  ↩
                                             fps.

Sound:

 * On/Off - Turns sound On/Off.
```

1.11 AmiDog's Movie Player - Menus - Output

```
 File   General   Output   MPEG   Misc


ColorMode:

 * Gray  (4/6/8 bit) - 16/64/256 colors, gray display.
 * Color (8 bit)     - 256 colors, ordered dither.
 * HAM   (6/8 bit)   - HAM6 or HAM8.

Gray depth:

 * 4/6/8 - 16/64/256 gray colors, lower depth is faster due to less CHIP accesses ↩
    .

HAM Depth:

 * 6/8 - HAM6 (4096 colors) or HAM8 (262144 colors).

HAM Width:

 * 1/2/4 - The amount of HAM pixels per RGB pixel.

HAM Quality:

 * Normal/High - Only affects 1/2-width, biggest difference on 1-width.

HAM S-Lores:

 * On/Off - If On, AMP will use SuperLores (PAL only) for MPEGs up to 160*128.

CGFX Depth:

 * 8/15,16/24,32 - The depth used for Best Mode screenmode, ignored on AGA.
```

For 15,16 and 24,32, AMP will try to use the highest, that is, if your gfx board support both 15 and 16bit, 16bit will be used.

Size:

* 50%/100%/200%/Full screen - The size at which the MPEG is played.

NOTE: On gfx boards the minimum depth is 8 (automatically adjusted if required).

1.12 AmiDog's Movie Player - Menus - Output - HAM Width

Back

To understand this you must first know the basics about how HAM modes work. I'm not an expert on this, and just before christmas last year (1998), I didn't know much at all. Anyway, here goes.

On a HAM screen a pixel is either one from the 16/64 colors in the palette or a a "finetuned" version of the pixel next to the left, that is, you can modify one of the three color components Red, Green, or Blue (RGB). Two bits per pixel is used to determine if "finetune" or palette color should be used. That is why a HAM6 screen is 6bit but only gives you 16 colors (4bit).

This means that to get one specific RGB value, you must either make sure that it's one of the 16/64 colors in the palette, or you must use three (3) pixels on screen to achieve the correct RGB values. This is why AMP supports different HAM widths.

As you might know, there are no 3-width screenmodes available, for example, there are 320*256 (1-width), 640*256 (2-width) and 1280*256 (4-width) for PAL. So to be able to get the right RGB value, you must use a screen which is four (4) times as wide as it is high. Since you then will get the right RGB value, these screens are often callen 12bit/18bit since you get 4/6bit per RGB component and there are a total of three components per pixel (2^(4+4+4)=2^12=12bit, 2^(6+6+6)=2^18=18bit).

The major let down by using 4-width is that it only works on PAL/NTSC/HighGFX and that it's terribly slow. That's where the 1-width and 2-width modes comes in. They try to achieve the best possible quality with less pixels.

The 2-width uses a simple but very efficient theory about in which order the RGB components should be changed to get the best quality while only changing two of them per pixel.

The 1-width uses a similar theory (which I've invented myself) but ofcourse produces a worse quality, it also requires more computations per pixel and thus is slower, but since a PPC is very powerful, and the main bottleneck is the CHIP accesses, you will

actually get a faster playback using 1-width.

I hope the text above explains why certain HAM widths only work with certain MPEG sizes, it's for example impossible to use 2-width on anything bigger than 640*512 since there are no available screenmodes, and if you use a VGA monitor, you can't display any screenmode larger than 640*512 (overscan not counted) which means that 2-width is only supported for MPEGs up to 320*256.

1.13 AmiDog's Movie Player - Menus - Output - HAM Quality

Back

Please read the text about HAM Width first if not already done.

Below you can read what the difference is when using High instead of Normal Quality.

2-width - The difference in quality is very tiny here. What AMP does is that it takes in account which RGB components that most urgently need to be changed in order to (mathematically) give the best result. On some MPEGs this can actually reduce the quality.

1-width - Here the difference in quality is quite hughe. When using high quality on this mode, AMP will use a predefined palette as well as using the normal "finetuning". For each pixel, AMP will calculate something called " color distance" to determine if it should use one of the 16/64 colors in the palette, or "finetuning". This ofcourse makes it even slower, but the quality is much better, and this mode works on every monitor, including VGA.

NOTE: 1-width high quality is the only mode which uses the palette, all others rely on "finetuning" for displaying the image.

1.14 AmiDog's Movie Player - Menus - Output - HAM S-Lores

Back

The Amiga actually supports something called Super Lores which aren't used very often. It's simple the same as a Lores screenmode but with half the height, i.e. PAL Lores is 320*256, and PAL Super Lores is 320*128. I found out that Super Lores also work on Hires screens, ie PAL Hires Super Lores is 640*128.

```
By using these modes, AMP is able to display a MPEG up to 160*128 in fullscreen ↩
    using
2/4-width HAM since these modes requires a screen which is 2/4 times as wide as ↩
    it's
height. As a small bonus, these MPEGs will be played faster and in fullscreen.

The difference in speed between normal PAL Lores and PAL Super Lores, and which ↩
    is most
clearly visible in 4-width HAM, is due to less CHIP memory bandwidth being used ↩
    by the
custom chipset since the screen is smaller, this also gives an idea about how ↩
    slow AGA
is when using HAM8.

NOTE: Super Lores is only supported for PAL and 2/4-width HAM.
```

1.15 AmiDog's Movie Player - Menus - MPEG

```
 File   General   Output   MPEG   Misc


Interpolation:

 * On/Off - On gives a slightly better YUV->RGB quality but is way slower.

Color Quality:

 * Normal/High - High gives better quality on 4:2:0 interlaced MPEGs, all
                 other MPEGs are not affected. This only affect 8bit output.
```

1.16 AmiDog's Movie Player - Menus - MPEG - Interpolation

```
 Back


I will not get too technical here, but I hope that you all will have some idea ↩
    about
what Interpolation is when you've read this text.

First some information about why interpolation "is" nesessary for MPEGs.

MPEGs are not stored in RGB as many pictures formats do, but instead in YUV, a ↩
    way which
separates the contrast and color components, the Y value is alwasy the contrast ↩
    of the
pixel, while UV gives the color (this is not really true, but it makes it easier ↩
    for you
to understand).

A MPEG don't very often have both Y and UV values for all pixels, for example, ↩
    4:2:0 means
```

```
that Y is there for all pixel, but UV are only there for every other pixel in  ↩
    every other
row, look below.

YUV Y.. YUV Y..
Y.. Y.. Y.. Y..
YUV Y.. YUV Y..
Y.. Y.. Y.. Y..

This is where interpolation comes in. Interpolation is a way of "finding out"  ↩
    which value
the UV components should have had if they would have been supplied. This can be  ↩
    done in
several ways, by either just looking at the pixels next to the "missing" one, or  ↩
    by looking
at several pixels.

Using interpolation will require quite a lot of computations. Depending on how  ↩
    the MPEG is
stored, the amount will differ. A 4:2:2 MPEG only reuires one pass of  ↩
    interpolation per
frame (horizontal), while a 4:2:0 MPEG requires two passes (horizontal+vertical).

Using interpolation will ofcourse give a better quality when later doing the YUV ↩
    ->RGB
conversion which is required since very few gfx boards support YUV directly. The  ↩
    RGB
components will be used to display the image in HAM6/8 or 15/16/24/32bit.

Since the Y value is always there for every pixel, gray will always get a very  ↩
    good quality
and interpolation is only used when required (ie when doing YUV->RGB conversion).

Conclusion: Using interpolation will improve the quality, but since the Y value  ↩
    is there for
every pixel, the difference will not get very big, and when it comes to moving  ↩
    frames, perhaps
at 25fps, then your eyes aren't fast enough to tell the difference. Perhaps if I  ↩
    add the
possibility to save the decoded MPEG as some kind of ANIM format, it might be  ↩
    usefull.
```

1.17 AmiDog's Movie Player - Menus - Misc

```
 File   General   Output   MPEG   Misc


Statistics:

 * Normal   - Shows size of the MPEG and FPS.
 * Detailed - Show same as Normal plus screensize and chunkybuffer width (only  ↩
     for debugging).
 * Off      - Shows no statistics at all.
```

1.18 AmiDog's Movie Player - Speed

```
CGFX speed: (Measurements using v1.02, CGFX v3 and a BVision and a 240*176 MPEG)

 -   Color Mode  -    AMP    - IsisPPC -

 * 8bit (Gray)   : 50.8 fps. :   N/A.
 * 8bit (Color)  : 42.4 fps. : 27.9 fps.
 * 16bit (Color) : 37.3 fps. : 29.4 fps.
 * 24bit (Color) : 37.2 fps. : 28.9 fps.

AGA speed : (Measurements using v1.01, AGA in PAL Lores and a 240*176 MPEG)

 -   Color Mode  -    AMP    - IsisPPC -

 * 4bit (Gray)   : 40.2 fps. :   N/A.
 * 6bit (Gray)   : 37.3 fps. :   N/A.
 * 8bit (Gray)   : 33.8 fps. :   N/A.
 * 8bit (Color)  : 31.0 fps. : 12.9 fps.
 * HAM6 (1 Width): 21.6 fps. :   N/A.
 * HAM8 (1 Width): 20.4 fps. :   N/A.
 * HAM6 (2 Width): 22.1 fps. :   N/A.
 * HAM8 (2 Width): 19.4 fps. :   N/A.
 * HAM6 (4 Width): 20.3 fps. :   N/A.
 * HAM8 (4 Width): 12.2 fps. :   N/A.

 SuperLores speed: (Measurements using v1.01, AGA in PAL and a 160*120 MPEG)

 -   Color Mode  -    On     -   Off   -

 * HAM6 (2 Width): 41.6 fps. : 41.4 fps.
 * HAM8 (2 Width): 38.7 fps. : 37.1 fps.
 * HAM6 (4 Width): 38.1 fps. : 36.4 fps.
 * HAM8 (4 Width): 31.7 fps. : 22.8 fps.

NOTE1: For the HAM modes, Interpolation is Off and HAM Quality is Normal.

NOTE2: SuperLores PAL means a 640/320*128 screen, while a normal Lores is ↩
    640/320*256.

NOTE3: Of some reason, IsisPPC reports 163 frames the first time the 240*176 MPEG ↩
     is played,
       but 161 frames all of the following times, strange!

NOTE4: AMP is 2.40 times faster than IsisPPC on AGA using 8bit color and PAL.

The 240*176 MPEG is a Wallace and Gromit MPEG played from RAM.
The 160*120 MPEG is a Tintin MPEG played from RAM.

All tests are performed using an A1200T 603e'200, 040'25, 128MB and OS3.1 from ↩
    Workbench.
```

1.19 AmiDog's Movie Player - ShareWare

After your free trial period of 30 days, if you decide to keep using it,
please send $15, £10, 20DM, 100SEK or equal amount of any other currency to:

Mathias Roslund
Sveav. 2b, nb
S-702 14 Orebro
Sweden

Thanks!

NOTE: I need your name and e-mail address to be able to send you your keyfile!

1.20 AmiDog's Movie Player - Disclaimer

Remember! You use this piece of software at your own risk!
I can never be held responsible for any sort of damage caused
to your software or hardware by the use of this product!

Bugreports and suggestions might be sent to one of my addresses.

1.21 AmiDog's Movie Player - Development

This product has been developed totaly by me using EGCS/GCC.

Since I'm a student, I just don't have the time to spend
several hours a day developing this product, especially
when approaching Christmas and summer since I then will have
a lot of schoolwork to finish. Therefor please don't write
to me and complain about the slow development! Thanks!

AmiDog's Movie Player is developed using:

```
v0.00-0.31   Amiga1200HD -> 040/FPU/MMU'25, 603e'200, 2+32MB, AGA.
v0.40-0.50   A1300Ti     -> 040/FPU/MMU'25, 603e'200, 2+128MB, AGA.
v1.00-       A1300Ti     -> 040/FPU/MMU'25, 603e'200, 2+128MB, BVision.
```

1.22 AmiDog's Movie Player - Bugs

* None known.

There might be more bugs, so you use it at your own risk!

1.23 AmiDog's Movie Player - Contact

Bugreports, suggestions, comments or anything else you may
want to contact me about can preferably be sent by e-mail to:

amidog@amidog.com

You may however also contact me by normal mail:

Mathias Roslund
Sveav. 2b, nb
S-702 14 Orebro
Sweden

1.24 AmiDog's Movie Player - Thanks!

I would like to thank the following persons:

* Stefan Burström - For answering all my (stupid?) GCC PPC questions.
* Jesper Svennevid - For giving me the C-only C2P and helping me getting it to work
  and for writing the now obsolete sound routines.
* Mikael Kalms - For helping me getting the 4bit C2P to do only 4bit C2P.
* Steffen Haeuser - For always being very helpful.

And ofcourse all of you who have registered:

* Thomas Kölsch, Germany
* Ketil Jensen, Denmark
* Morgan Johansson, Sweden
* Frank Dietrich, Germany
* Lee Cook, England

(If something is wrong above, or if you want to get removed from the list, just write!)

1.25 AmiDog's Movie Player - Future

This is what I currently plan to add, it is NOT in priority order, and it might change without further notice!

* Custom playback size.
* Seeking in audio MPEGs with audio enabled.
* Seeking in FLI/FLC/AVI/QT anims.
* ANIM/... support.

1.26 AmiDog's Movie Player - History

v1.10 -990816
      -Finally some progress. I've completely rewritten the AsyncIOPPC part so that it now

```
 also does demuxing (splits the MPEG into a video and a audio "stream" on ↩
    the fly) which
 will make it possible to get synchronized audio and video playback. It ↩
    also means that
 the video and audio decoders will get separeated and thus can be exchanged ↩
     without
 problems. Currently seeking don't work, I'll try to fix it later.
-990817
-Spent several hours tracing a stupid bug in the new demuxing AsyncIOPPC ↩
   routines, it
 should now be 100% bug free (I hope).
-The demuxer is still not optimized and the audio routines are even worse, ↩
   but it's a
 start. Also, AMP only works with MPEG system streams (audio + video) ↩
    currently.
-I've spent some more time trying to find all bugs (if any) and I found one ↩
     related
 to the sound buffer, but I still don't know why the audio.device code fail ↩
     sometimes.
-990818
-AMP now supports video-only MPEGs once again. While adding this I came up ↩
   with better
 ideas of how to deal with some of the AsyncIOPPC stuff.
-Started commandline support, it works quite ok, but there is a bug ↩
   somewhere which
 I'll have to trace.
-AMP now recognizes a bunch of other fileformats, which might be usefull ↩
   one day.
-990818 (later)
-Finished the CLI support. It's not all that nice, but it works, and that's ↩
     probably
 what counts right now.
-Cleaned up some parts of the code (AsyncIOPPC mainly) and saved a few kb ↩
   of exe size.
-990819
-Cleaned up the internal buffer handling in the MPEG audio decoder, it was ↩
   really a
 waste of CPU power and memory! The new way is faster.
-I do get random crashes, well they actually seam to appear one out of four ↩
     times I
 play a MPEG, and it doesn't matter if sound is on or not.
-990820
-The frameskip is now working as before, I had changed it a little for dev ↩
   purposes.
-Some minor code cleanup, released a beta for people to try out.
-990822
-AMP didn't recognize MPEG streams with some "junk" in their header (i.e. ↩
   before the
 first startcode). This is now fixed.
-Hmm, just for your information. AMP with current soundcode on = 17fps ( ↩
   many frames
 skipped), soundcode off = 78fps (no frames skipped). AMP with soundcode ↩
    off and
 no frameskip while mpg123 was playing the MPEG audio, 50fps... I guess ↩
    that AMP can
 be MUCH faster with new audio code!
-990822/23
```

```
      -Well, it's actually another day now, but what the heck.
      -I've now replaced the audio routines, and guess what!? My first measurement with the
       same MPEG as above gave me 62fps, and that was WITHOUT any optimizations! So, after
       enabling, I now get 70fps, that is, only 8fps lower than without sound! However,
       this is in 22050 Hz 8bit mono. The MPEG audio decoder lets me choose a lot of different
       sound modes, so later on you'll be able to get 44100Hz 16bit stereo if you like (and
       have the hardware) that is.
      -The sad thing is that I've now added some bugs somewhere which means that I'll have
       to reboot my computer after each play, otherwise AMP will just hang the second time
       I use it, I've no idea about why yet.
      -990823
      -The bug which made AMP crash the second time it was started has been removed, it now
       hasn't failed on me a single time actually :)
      -Fixed so that AMP exits nicely if the Demuxer can't open the file. Currently this
       happends if one don't use an absolute path (i.e. movie.mpg instead of Work:movie.mpg
       even if you've got AMP in that directory).
      -Moved the opening of the file from the Demux task to the main task, this way AMP can
       now open files with relative paths again.
      -For those who actually read this, use the GUI with caution, after adding the CLI support
       I've hardly used the GUI at all, so I have no idea if it works ok or not.

v1.20 -990824
      -Added FLI/FLC support, it's still very slow due to many disk accesses, and not all kind
       of "chunks" are supported.
      -Small MPEGs (<256k) would make AMP hang since the buffers never got fully filled, fixed.
      -Added a general version of my asyncio routine, this is usefull for FLI/FLC for example.
      -Started rewriting some AsyncIO/Demuxer task stuff to avoid some strange situations which
       might crash AMP.
      -The FLI/FLC player now uses the new AsyncIO routines.
      -990825
      -Changed the way the second Demux/AsyncIO task is handled to be fully system friendly.
       That is, instead of just deleting the task, I'm signaling it to quit and I'll wait until
       the task has quit properly BEFORE I free the semaphore/signals. As things were before, in
       som special cases, I might have freed the signals/semaphore and used them after that, I
       might also have signaled the Demux/AsyncIO task after it hand quit using an obsolete task
```

```
 pointer. All these things didn't give any problems until I added the FLI/ ↩
    FLC stuff and the
 new general AsyncIO routines. These changes should hopefully remove the  ↩
    last reasons for
 why AMP might have crashed seamingly random.
-The internal MPEG audio buffer didn't handle EOF properly, fixed.
-The timer will now get reset AFTER the sound has been intialized,  ↩
   otherwise AMP will have
 to skip a lot of frames in the beginning to make up for the delay. The  ↩
    MPEG I tried it
 with skipped none instead of 120 frames in the first 10 seconds.
-Bugfixed the FLI/FLC routines, there were many serious bugs. I've also  ↩
   optimized the
 routines to get rid of some bottlenecks.
-990826
-Started adding AVI/QT support. Currently the QT parser is very early, the  ↩
   AVI parser
 is however working quite well (I guess). No codec is supported yet.
-990827
-Added support for 'Microsoft Video 1' (8/16 bit) codec (AVI).
-990828
-After spending half the day tracing bugs, I've added support for the CVID  ↩
   codec (AVI).
-990830
-After having AMP as shareware for half a year, and not getting a single  ↩
   registration, I've
 decided to make it crippled, so I've created some code for handling  ↩
    keyfiles.
-Started working on a QT parser, still a lot of work to be done.
-990831
-Found a bug int the FLI/FLC parser, made a workaround which hopefully  ↩
   should work. I
 also added an "emergency exit" in case the file is seriously currupted.
-Added keyfile code to AMP, you wont be able to use FLI/FLC/AVI without a  ↩
   keyfile.
-Some changes to the AVI parser. One more codec added, YUY2 (AVI).
-990901
-I got my first two registrations today :) Thanks a lot guys!
-Rewrote most of the AsyncIO code to be more general and to support seeking ↩
   .
-Increased the AsyncIO security (i.e. error checking) and made the required ↩
    modifications
 to the FLI/FLC plugin. Support for the ftell command has also been added.
-The AVI parser now uses AsyncIO and has been changed to be way more secure ↩
   , that is,
 AMP wont crash (hopefully) even if the AVI is cropped or seriously damaged ↩
    .
-Filenames with spaces should now work.
-Cleaned up the exit code used when you get the "AMP exited due to some  ↩
   error" message,
 it's still far from perfect and a reboot is strongly recommended.
-The QT parser finally let me decode and display some frames! There is  ↩
   still a lot of
 work needed before it can be added to AMP.
-990902
-Found a bug in the AsyncIO routines, the value returned from the read  ↩
   function when
```

```
 the last byte(s) was read was one too low.
-The MPEG code was changed so that the demuxer only is used when required, ↩
   that is, not
 for video-only streams. Removed the video-only MPEG support in the demuxer ↩
     routines.
-Seeking now works again, but ONLY on video-only streams, simply turning  ↩
   off the audio
 won't work currently.
-990903
-More work on the QT support, I'm actually getting something displayed now  ↩
   :)
-990904
-Yet more QT work. It appears to be quite stable now. A JPEG decoder has  ↩
   been added,
 but it seams to be a bit buggy, the gfx looks a bit strange in places.
-The first AVI audio decoder has been added, it's a 8/16bit PCM one and it  ↩
   seams to
 work quite ok.
-990905
-Some changes to the AsyncIO routines, it's now possible to continue using  ↩
   the routines
 after an EOF has occured since the "buffer filler" wont quit when this  ↩
    happends.
-I currently have some problems with the AsyncIO, sometimes when the  ↩
   routines are started,
 it seams to fill the buffer with only zeros, and thus the FLI/FLC/AVI  ↩
    players will quit.
-Major AVI code cleanup, I've removed many kb of unused code. The audio  ↩
   code has also
 been improved a lot, and there is not much to do before it's finished.
-990906
-All exit calls in the MPEG routines has now been replaced with a call to a ↩
    function which
 will stop the AsyncIO/Demuxer routines and there after close audio/screen  ↩
    etc. This should
 (hopefully) make AMP exit more nicely when a problem occurs. This way of " ↩
    hacking" is not
 required for FLI/FLC/AVI etc. since those are written from scratch by me,  ↩
    and therefor are
 written to exit very cleanly after doing a full cleanup, and thus AMP will ↩
     not have to exit
 even if some serious errors occur.
-The AVI audio stuff now supports all fps rates and audio rates (below  ↩
   28000 Hz) so there
 is now very little to finish before making a new release of AMP.
-990908
-Fixed the AsyncIO bug I wrote about above, so it should now be ok (I hope) ↩
   .
-The image will now get centered when using FLI/FLC/AVI. There is however  ↩
   still no AGA
 support for those filetypes since I still need to rewrite some parts of  ↩
    the AmigaOS
 routines. The FLI/FLC/AVI player also uses CGFX for displaying (=slow).
-Some other minor changes as well.
-Fixed one AVI audio bug, of some reason the fps got one to low (overflow?) ↩
    which meant
 that too much audio was played each frame.
```

```
      -Made the same changes to the demuxer that fixed my AsyncIO problems, ↩
         although the demuxer
       hasn't failed on me yet, but who knows, I might have solved someone elses ↩
          problems?
      -990909
      -Added some error messages when the best mode id failed (CGFX).
      -AVI files can now be played in any depth (8bit color/gray, 15/16/24 or 32 ↩
         bit).
      -FLI/FLC/AVI should now work on AGA, I've not tested it yet though. I'll do ↩
          tomorrow.
      -990910
      -Tested the FLI/FLC/AVI AGA support and it works great :) I just watched " ↩
         Aqua-Barbie Girl"
       (15fps AVI) using 2-width HAM8 on my TV, great quality!
      -The Limit FPS setting now also work on FLI/FLC/AVI.
      -FLI/FLC can now be viewed in any color depth, 8/15/16/24/32 bit (CGFX only ↩
         ). This is
       however not recommended since it's very slow.
      -Found a bug when using Limit FPS and AGA, removed. I've also figured out ↩
         how to calculate
       the FLI/FLC fps, so those are now played in the right speed!

v1.21 -990911
      -Added a version string to AMP, so now "version amp" will work.
      -Removed an AsyncIO bug, SEEK_SET didn't work properly.
      -FLI/FLC/AVI was always played at their right frame rate, even if the user ↩
         choosed another
       one, fixed.
      -Since there is no 4/6bit gray support for FLI/FLC/AVI, 8bit is currently ↩
         forced to
       avoid strange effects. Also, FLI/FLC can't be forced into gray, they are ↩
          always color.
      -If the MPEG had a framerate of 24 fps the audio would get out of sync due ↩
         to the fact that
       22050/24 = 918.75 which would get rounded to 918 and thus 18 bytes too ↩
          little would get
       played each second. Added a resync which should avoid this problem.
      -Removed another seeking related AsyncIO bug, and made some optimizations ↩
         while I was at it.
      -Started adding the QT stuff to the main AMP exe, it's working quite ok, ↩
         although not perfectly.
      -The 32bit->15/16bit conversion routines didn't work properly, fixed.
      -990912
      -With audio off, seeking now works on MPEG system streams (video+audio) as ↩
         well.
      -Added one more codec for QT, Apple Video (RPZA). The QT stuff is still a ↩
         bit unstable
       (atleast with streams with 2 or more tracks), don't know why though.
      -The QT's can now be played at the right frame rate (atleast single-track ↩
         ones).

v1.22 -990914
      -Started adding window support just to find out how slow CGFX really is ↩
         when you don't
       have the same screen depth as anim depth. And since CGFX neither support ↩
          15bit, nor
```

```
 16bit but only 24/32bit input data, I made a hack to get some more speed.  ↩
    It's probably
 a stupid hack that people will have problems with, so it will be optional. ↩
     But hey,
 I increased the speed from 10fps to 25fps with the AVI I tried it with (on ↩
     16bit WB).
-990916
-The window support now works (or should work) on any Workbench with a  ↩
   depth of atleast
 15bit. The speed hack is now optional, and this will be for screen replay  ↩
    as well.
-The window support now works with MPEG as well, and it's probably finished ↩
    unless some
 bug is hiding somewhere. Using window and speed hack should be done with  ↩
    care, otherwise
 you might get parts of the display all over Workbench.
-The FLI/FLC 8bit->32bit converter is now about 15% faster.
-990917
-Just some minor changes before making the release, hope the window stuff  ↩
   work ok now.
-Erhm, when the MPEG was stopped and audio was used, the MPEG audio stuff  ↩
   would not get
 reset properly, and audio.device wasn't closed. This has been fixed.

v1.23 -990919
-AMP now decodes less audio each time, this should lower the overhead when  ↩
   audio decoding
 is required and thus make the playback smoother. I've also made some  ↩
    changes to the audio
 buffer (Demuxer) handling.
-Improved the video buffer handling (Demuxer) which makes AMP about 8%  ↩
   faster when using
 MPEG system streams.
-A new AVI codec added, Microsoft RLE8 (8 bit), and this one actually works ↩
    with full
 optimizations :) Guess I'll have to figure out why most of the others fail ↩
    .
-Yeah! I found the bug which made most decoders fail if optimization were  ↩
   used, the
 strange thing is that there is nothing wrong with the code itself, but of  ↩
    some reason
 GCC/EGCS (both versions I've tried) makes a mess out of it when optimizing ↩
    . The size
 of AMP got reduced from 780k to 670k thanks to this.
-990921
-Made some changes to the timer, it should no longer give strange results  ↩
   when watching
 large movies.
-I'm now feeding audio.device with 8 times as much data each time. With a  ↩
   MPEG that uses
 24fps, this means 3 instead of 24 PPC<->audio.device contextswitches/ ↩
    feedings each second,
 this makes AMP quite a bit faster. The problem is that I have to limit the ↩
     fps in between
 the feedings in the same way as when audio isn't used, and this can give  ↩
    audio interruption
 in some cases.
```

```
-Increase the amount of audio that is decoded each time, it's still only half of what it
 use to be, also, I now managed to get my first "Audio buffer low" message, so now if this
 ever happends, AMP will just stop the audio and continue playing. I then doubled the buffer
 used by the deumxer, it's now 512KB, and the "Audio buffer low" message disappeared, I
 hope that this buffer size is big enough. Please report eny problems.
-By the way, I just watched a music video (352x288 25fps) in a window on my 16bit BVision
 Workbench with perfect audio (using 603e'200). Ok many B-frames were skipped, but anyway.
```