Default

COLLABORATORS			
	TITLE : Default		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		December 25, 2022	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

Default

1.1 The Games Master System V0.3B

```
                         T H E   G A M E S   M A S T E R   S Y S T E M

                         VERSION 0.3B

                      GENERAL DOCUMENTATION
```

I've only just started writing this doc so it is quite incomplete. I will get more of it done in time for the next release.

1.2 Introduction

This introductory text is taken directly from the web pages:

http://www.compkarori.co.nz/amiga/gameslib/

WHAT IS THE GAMES MASTER SYSTEM?

The Games Master System is a solution to one of the biggest problems the Amiga community has ever faced. What problem? Games support in the OS! Windows has it, the Apple has it, and the poor old Amiga has none. This lack of support has brought up some quite serious problems when it comes to hardware compatibility in games. Every Amiga owner encounters them at some time, if not often. There have been efforts made towards libraries specifically tailored to handling graphics cards, but these are missing the big picture.

THE BIG PICTURE

Hardware bashing is fast, sometimes many times as fast as an equivalent system legal routine. However hardware bashing is not very compatible with other hardware types, resulting in unpredictable results on other systems. No programmer can be expected to support dozens of obscure hardware configurations... just look at what it did to the PC, you can't install a game without going through and setting IRQs for sound cards. Let's face it, if the problem isn't dealt with directly, then the situation is just going to get worse as new hardware is released.

So how are we going to deal with this ever increasing problem? No-one has ever come up with an acceptable solution for our little Amiga. Maybe we are getting close to retargettable graphics, but what about retargettable sound? What about user support? What about networking? Where's our support for all those different joysticks? Where's the real support for writing games in an OS? What about all the other stuff people keep moaning about? Well after a lot of design and investigation into this, a solution is already on its way. It's a clever little system that everyone will come to know as...

THE GAMES MASTER SYSTEM

The "Games Master System" solves all the problems developers have faced in the past, and are still facing today. Project GMS encompasses not only the games.library but also all the GPI's, documentation, developer information, and most importantly the user prefences program (GMS Prefs). All these things have been designed to work together and will achieve the following:

* Erradication of the need to bash the hardware from within games.
* Make it easier to migrate from the current Amigas to the new Power Amigas.
* Make games programming, easier, faster, and more productive.
* Give users the ability to modify any game to suit their requirements.

HOW DOES IT WORK?

First lets look at the way games are written on the Amiga today. A lot of experienced coders would tell you that a lot of their source is automated, put into macros, drawing routines, sound routines, and so it goes on. Rather interestingly, this is usually the same stuff that is doing all the hardware hacking.

By taking these commonly used routines and putting them into a library we remove the hardware compatibility problem immediately. Unfortunately for us, although it works this method is just not fast enough for the speed that games require. Enter the use of GPI's!

A GPI (short for "Games Programming Interface") is a collection of library functions specifically designed to perform a task for just one hardware device. Each GPI can therefore be built to do whatever it wants without having to worry about compatibility problems. New versions of an existing GPI can be written for different hardware devices, so graphics cards, sound cards, 3D chips etc can be supported.

The benefit of this is that the user gets the best possible speed, while the programmer can simply utilise the built-in routines and know that not only will their game be fast, but also be compatible with Amigas everywhere. Cool!

USER SUPPORT

If you're a games player then there have probably been a number of times where you've thought "Why can't I use my Sega Joypad?", "I want to run in a DBLPAL screen!", "I want multi-tasking!", "I want to turn the music off!", "I want, I want, I want!". Unfortunately, that's what it sounds like to developers who don't have time to support all these things!

Luckily an important feature of the Games Master System is the level of support given to the user. The GMS Prefs program allows the user to select levels of mode promotion, type of joystick used, vector detail, networking, C2P routines, music re-direction, task priorities, and much much more. This solves a lot of the moans and gripes that users have had in the past, and since this is all transparent to the programmer, user support is easily achieved. Hopefully this news will make you all very happy!

ANYTHING ELSE?

Well that just about sums up the Games Master System rather well. Look around the rest of the site to get more detail on the things mentioned here, and other things that we haven't gone into yet. The GMS binaries and documents are available on Aminet, in dev/misc/GMSV03B.lha. Remember to ask us if you have any questions about the project!

1.3 OverView

OVERVIEW OF THE GAMES MASTER SYSTEM

Project GMS started in the beginning of April 1996, in an effort to provide games support in the Amiga OS. The overall aim is to write the best games interface we possibly can, which should eventuate into a system that everyone can enjoy. The prime objectives of GMS are:

1. To erradicate the need to bash the hardware from within games.
2. To make it easier to migrate from the current Amigas to the new Power Amigas.
3. To make games programming easier, faster, and more productive.

GMS has been designed to be fully extendible in ways that will make future improvements very easy to implement. The system is split into a number of sub-sections: The master library, the GPI's, the debugger, and the preferences program. This is further enhanced by identifiable data structures, which allow us to write new structures in future, without overhauling the functions.

The library itself serves as a "skeleton" of function calls, rather than having all the code residing within the library itself. The library is "filled out" on opening by the use of Games Programming Interfaces (GPI's). A GPI consists of a group of function calls pertaining to one particular area of games programming. For example, the Screens.GPI contains routines for opening and closing a screen, altering the screen's palette, and placing sprites onto a screen. Other GPI's are purely optional, and must be initialised before use. This saves a lot of memory as you only have to load in what you need.

GPI's can be written and re-written for any hardware setup imaginable. It is possible to split them up into even smaller files if desirable. An example of this is the screens.gpi's Chunky 2 Planar routines, which can give the user a variety of options for his system.

To see how everything links together, view the Communication.iff file to see a graphic representation of the links and connections now.

There will of course be a version of the games.library for the new Amiga's, so an AGA game written now could still work on these new machines! The great thing is that this library really closes the gap between differences in machine architecture. Even games intended for a PPC could work on an Amiga with CyberGfx card, granted that you compiled a 680x0 conversion.

In the future I hope to have available:

* Support for graphics cards.
* Support for sound cards.
* Vector and 3D functions.
* Support for all the new hardware capabilities in PPC Amigas.

This is looking quite some time into the future of course, but GMS is designed for these things so it's all quite possible.

1.4 I want to write a really awesome-cool-funky-thingofa GPI.

WRITING NEW GPI'S

Anyone can write a GPI, but the first thing you must do is email me about your idea and describe what it will do. I will then consider wether or not a whole new GPI should be created, or if your idea should be added to an existing GPI. It may turn out that someone else has the same idea as you, in which case I will put you in contact with that person.

If in the event a whole new GPI needs to be created, I will first ask you to list all the possible functions you think will be needed. Once that is figured out we will design the structures that will be required - usually there will only be one of these, so it’s important that we get it right.

You will then be sent a GPI development kit which details how to integrate your GPI with the games.library. GPI’s are based on normal system libraries, but have the addition of some code supporting function remapping. Later when you have written the GPI you have to send it to me and I will suggest any changes to the functions etc. Once everything is okay, then it’s an official GPI.

The only other condition that you must adhere to is that your GPI is written in assembler, or very good C. Also, it helps to send me the souuce at some point so that if you lost interest later on, I could still update the GPI when necessary.

Remember there may be a lot of people using your GPI so I must ensure that it’s 100% OK and can be upgraded for future Amigas.

1.5 Really cool features!

CURRENT FEATURES OF THE GAMES.LIBRARY

These are just some of the features that have so far been implemented. For the complete low-down on all the features check the developer information files.

* Full sprite support, and that is: All available sprite dimensions, colour table offsets, 16 colour sprites, width-doubled sprites, full animation support, lo-res, hi-res, and superhi-res support,

* The best sound support to date, including: Support for sound priorities, intelligent dynamic channel play-back, channel modulation for special effects, IFF and unpacking support for crunched sounds.

* Screen fading of palettes to any colour, palette morphing, and fast fading to black and white colours.

* Full support for raster/copperlists, with effects such as: ColourLists, Mirror, Flood, Screen and Sprite Splitting, and Palette Changes.

* Allows you to support all different kinds of input devices (joysticks, joypads, mouse etc) through just one simple function call. This enables you to support devices that don’t even exist yet.

* User preferences program to allow full configuration of a game’s functionality. This includes configuration for: Game/Task Priorities, Choice of networking, Mode Promotion, Joystick Config, Music

Redirection, and more.

* Support for all kinds of screen modes and resolutions, including VGA (scan-doubled) screens.

* Memory protection - Secure memory allocation and a freemem routine that will not crash your machine if you have written over your memory boundaries. You will be given a clear warning telling you exactly what went wrong (no guru numbers). You do not require an MMU for this feature.

* Smart Saving and Loading of files, with automatic packing and depacking. Packer support covers files crunched with XPK (external), PowerPacker (internal), and RNC methods 1&2 (internal).

* All games can multi-task with no drop in speed or performance.

FEATURES IN PROGRESS

* A Debug.GPI that can output game data across a serial connection or to a custom window as the game runs. Will also include memory display, search and editing and possibly a disassembler all in one interface.

* Chunky 2 Planar that won't bother wasting time with conversions or copying if chunky mode is available in the hardware. Various routines for different systems will be user selectable.

* Networking support across modems, null modems, and TCP/IP.

* The Reko.GPI by Gerardo Iula, for the use of REKO cardsets. Also he is working on the Anim.GPI for animation effects.

* The Vector.GPI (saving the best till last). Will have absolutely everything, such as texture mapping, various shading, shadows, etc. The level of detail and the effects used will be entirely user selectable.

1.6 About Structures and Lists

STRUCTURES AND LISTS

One of the things you will notice about games.library structures is the version header in the first longword (eg "GSV1" for GameScreens). In the past, header fields have only ever been used for easily identifying files. If we didn't have them, we'd never know what sort of data we were dealing with.

Now the idea of these headers has now been taken and is being used in GMS structures. Why? Well for debugging, support for the growth of structures, and identifying exactly what any structure is used for. Also, it allows the functions to do different things according to what sort of structures you give them.

An example of this identification is for lists. What's a list? Well in the case of GMS a list is intended for processing 2 or more structures inside a function. This is the fastest way that you can process a whole lot of structures without having to make heaps of function calls. Lets say you wanted to load in 10 sounds from your hard-drive using InitSound(). Normally InitSound() takes a Sound Structure, but it can also identify a List by checking the header ID.

To illustrate, a typical list for initailsing/loading sounds looks like this:

```
SoundList:
  dc.l  "LIST"             ;List identification header.
  dc.l  SND_Boom           ;Pointers to each sound to load and
  dc.l  SND_Crash          ; initialise.
  dc.l  SND_Bang
  dc.l  SND_Ping
  dc.l  SND_Zoom
  dc.l  SND_Zig
  dc.l  SND_Zag
  dc.l  SND_Wang
  dc.l  SND_Whump
  dc.l  SND_Bong
  dc.l  LISTEND            ;Indicate an end to the list.
```

When you want to load all your sounds in, just use this piece of code:

```
  move.l  GMS_Base(pc),a6
  lea SoundList(pc),a0    ;a0 = Pointer to the soundlist.
  CALL  InitSound
  tst.l d0
  bne.s .error
```

Pretty easy right? Of course, there are lots of other functions that support lists. The not-so obvious ones are:

```
  Init_BOB()
  Init_Sprite()
  InitSound()
  FreeSound()
```

Some functions are specially written to be given lists only, eg Blit_BOBList(). This is mainly for speed reasons, as we don't want to waste time checking if a structure is a list or not in time critical situations.

That's basically the summary on lists. You may be interested to know that the GMS package is the only programmers aid that supports structures in this way. You will learn more about lists and how ID fields will help you in other areas of this doc.

1.7 Hints and Tips

GAMES MASTER SYSTEM

HINTS AND TIPS

This section is written to offer some friendly advice and tips on how to get full use from the games.library, and what tricks you can use to make sure your game runs at the highest speed possible. I'm still writing this section, but if you have a trick of your own that should be here, please write to me at sandman@welly.gen.nz.

1.1 GENERAL CODING TIPS

Less... equals More!

Never call the same routine twice in your main loop unless absolutely necessary. For example, look at this routine that calls Read_Key() twice:

```
----
Loop: lea KeyStruct(pc),a1
  CALL  Read_Key
  cmp.b #K_ESC,d0
  beq Game_Over

  ...
  Rest of main loop
  ...

  lea KeyStruct(pc),a1
  CALL  Read_Key
  cmp.b #" ",d0
  beq .Exit
  lea GameScreen(pc),a0
  CALL  Wait_OSVBL
  bra.s Loop

KeyStruct:
  ds.b  KP_SIZEOF

----
```

Do this instead...

```
Loop: lea KeyStruct(pc),a1
  CALL  Read_Key
  cmp.b #K_ESC,d0
  beq Game_Over

  ...
  Rest of main loop
  ...

  lea KeyStruct(pc),a1
  cmp.b #" ",KP_Key1(a1)
  beq .Exit
  lea GameScreen(pc),a0
  CALL  Wait_OSVBL
  bra.s Loop
```

```
KeyStruct:
  ds.b  KP_SIZEOF
```

As you can see the second version is faster because it doesn't make an extra call to Read_Key. Simple really, and this common sense applies to many situations.

1.2 MULTI-TASKING UNDER THE GAMES.LIBRARY

It is up to you wether you want to multi-task or not, I can only encourage you to do so. The games.library offers some special features to allow your game to multi-task effectively without disabling the OS entirely. Two functions are specifically related to multi-tasking, and they are:

```
  SetUserPri()
  ReturnToOS()
  AutoOSReturn()
  Wait_OSVBL()
```

Firstly, SetUserPri is a routine to set the priority of your task to a user specified level. The default setting is quite high (four) which will give your game a lot of system time. A setting of 8 is roughly the equivalent of a forbid()/permit(). It is important to let the user set the priority in GMS Prefs because as CPU's get faster, your game will need less CPU time. So try and make use of it just for that extra user-friendliness.

If for some reason you feel that you must still turn off multi-tasking, the least you should do is use screen switching, supported by ReturnToOS(). This is a very important function to use for true multi-tasking. Lets say you've opened the screen and now the view is taken over. But, what if the user wants to return to the OS momentarily without quitting the game? By calling ReturnToOS() the screen will leave the display and either a) drop out to a window on workbench or b) drop out to a standard OS screen. What will happen exactly is up to the user. The OS will now be available to the user until he signals the games.library to return to your task.

This function is further supported by the AutoOSReturn() function. This is very similar to ReturnToOS(), but checks the Amiga-M key combination for you and if found, switches the screen automatically. It's a lot easier to incorporate this in your main routine but some may prefer the ReturnToOS() function for better task control.

Finally is Wait_OSVBL(), which is probably the function you will use most often. This will perform an Amiga-M check before the vertical blank occurs, which can be very convenient in all circumstances. If you use this function whenever you have a VBL wait, your game should be 100% supportive of screen-switching without you needing to think much about it.

As a final note, whenver the OS is returned your game will be paused until the user enables you again. This is quite convenient, since we don't want the player dying when he can't see the action.

1.3 <Oh, I seem to have run out of room>

1.8 games.library/Sound Overview

SOUND.GPI OVERVIEW

The GPI for sound support is one of the most, (perhaps the best) interface for the support of Amiga sound effects. It features full sound prioritisation, intelligent channel selection on playback, and will eventually support special sound formats such as the common PC WAVE.

To illustrate the power of the Sound.GPI, here is an example. Lets say you develop your game on your A1200 to make dynamic use for all 4 sound channels. This is simply done by specifying CHANNEL_ALL in the SAM_Channel field. What that does is play the sound through whatever channel is currently available. Used in conjunction with the PlaySoundPri() function, you can make maximum use of all 4 channels, rather than just one channel with no prioritisation.

Clever right?

Sure, but if your game was to be run on an Amiga with say... 12 channels... then all those 12 channels will suddenly be supported by your game! This is because the Games Master System has been designed to allow your game to improve as the hardware gets better. Our aim is to get games written in 1997 to still have up-to-date sound support in 2007. (Yes, really).

As a further example, if you were to use IFF sounds loaded in from disk, you could support 16 bit sound, if the user was to update the sound files. (The hardware would have to support 16 bit sound of course). Packed sounds are also supported by the GPI so there are no problems in that department.

If you have any ideas for further improvements to the Sound.GPI, send them on in...

1.9 Screens Overview

SCREENS.GPI OVERVIEW

<Or maybe not>

1.10 Compatibility Problems

COMPATIBILITY PROBLEMS

One of the most important decisions I made in the design of GMS, was to get the absolute most out of what the Amiga hardware is capable of. The fact is, if I wrote GMS with respect to other gfx cards, there would be no:

```
  Sprites
  Hardware Scrolling
  Overscan
  Double Playfields
  Split Screens
  and RasterLists
```

Strangely enough, isn’t this what makes the Amiga unique? Also if the new Amiga’s came out with quadruple 256 colour playfields and 512 colour sprites, should I support that if other gfx boards don’t? Why should I not support it?

Well one of the goals of GMS, is to always be as up to date as the hardware that is available at the time. It’s vital if the Amiga platform is going to beat the competition, and there is a lot of it.

Now, this is at the cost of compabitility. How compatible you want your game to be on other systems is entirely your own choice. Generally, the more hardware-specific features you use, the more you risk your games failure on different hardware. If you use less, your game has an excellent chance of successfully working on all systems. Whatever happens, you will have to make your own decisions on the compatibility issue.

What I will do in this section is to try and help you face these problems, and hopefully overcome them. With some intelligent programming, you can still use features like sprites, rasterlists and hardware scrolling, and still keep your game running on other systems. It takes a little work but the least it will do is make a lot of people very happy. Good luck!

1.11 The Authors

THE AUTHORS

The Games Master System is written in assembler by Paul Manias (that’s me!). Paul has 4 years 68000 and games programming experience, and another 1 year in other languages like C and Pascal. Paul’s favourite past-times are sitting down, sleeping on glass shards, and eating soft toys. So far he has written two games of his own and contributed graphics to two others. None of those games have beeen released yet, for all sorts of various reasons. Luckily this is not the case with GMS.

GMSPrefs is written in E, by Richard Clark. Richard’s favourite past-times are standing, sending morse code via blinking, and talking to suspicious items of furniture.

Thanks to Jyrki Saarinen and Fabio Bizzetti for their donations to the project. Also to the many people that sent in ideas when the project first started (but we still need more!).

The web page exists thanks to Graeme Chiu, who owns a computer shop at CompKarori. To see the pages, visit:

http://www.compkarori.co.nz/amiga/gamelib/