A MOD Player for GBA

Bachelor’s Thesis in Computer Engineering

Lin Ge & Danqing Ni
A MOD Player for GBA

School of Information Science, Computer and Electrical Engineering
Halmstad University
Box 823, S-301 18 Halmstad, Sweden
June 2010
Acknowledgement

There is no royal road to learning…

--Anthony Trollope

We would like to thank LYPSON, a company from Halmstad, for giving us the opportunity to work with this project. Its trust and guidance since the beginning of the project give us a lot of help and confidence; and our supervisor, Edison Pignaton de Freitas, from Halmstad University, for his wonderful supervision and feedback during our thesis, especially about academic issues and our weekly report. Furthermore, we would also like to thank to Kenneth Nilsson. When we choose and write our thesis topic, he gave us a lot of helpful advice.

At last, we would like to thank them all other classmates and teachers for their valuable ideas and discussions regarding the project, and reviewing our thesis.
## Details

<table>
<thead>
<tr>
<th>Authors</th>
<th>Lin Ge &amp; Danqing Ni</th>
</tr>
</thead>
<tbody>
<tr>
<td>University</td>
<td>Halmstad University, Sweden</td>
</tr>
<tr>
<td>Degree Program</td>
<td>Computer Science and Engineering</td>
</tr>
<tr>
<td>Title of Thesis</td>
<td>A MOD Player for GBA</td>
</tr>
<tr>
<td>Supervisor</td>
<td>Edison Pignaton de Freitas</td>
</tr>
</tbody>
</table>
Abstract

This bachelor thesis describes the development of a MOD music player to run on GBA (Game Boy Advance) for Lypson Game Engine. GBA is a basic platform for embedded development, and the stereo system makes it possible to use the platform as a music player. The sound players of the GBA for Lypson Game Engine were designed to play wave files and the size of typical wave files is in the order of megabytes. MOD files are much smaller when compared with wave ones. Therefore, to avoid consuming the resources of the CPU and memory to process and store wave files, the use of MOD files represents a better alternative.

The development took C++ as programming language and the development platform used was HAM. The first step was to obtain samples of music, and then control the hardware of GBA to play sound. After that, came the phase of combining it with Lipson Game Engine. These tasks enabled the acquisition of knowledge about the frame of MOD files; learning how to make the GBA play sound and mastering the operating instruction of GBA hardware by the process of development. In addition, it provided a chance to learn about embedded development, which represented a starting point to learn about embedded programming in general.

As for the main result, it was achieved by the successful development of the MOD Player, which is now running on the Lypson Game Engine. As the MOD files are of small size, the music player is more efficient when compared with those previously used.

Key words: GBA, Lypson Game Engine, MOD Player, Embedded C++ Development
Contents

ACKNOWLEDGEMENT ........................................................................................................I
DETAILS ..........................................................................................................................II
ABSTRACT ....................................................................................................................III
CONTENTS .....................................................................................................................IV
TABLE............................................................................................................................VI
FIGURE.........................................................................................................................VI
LISTING ..........................................................................................................................VII

I. INTRODUCTION ........................................................................................................1
   1.1 BACKGROUND .................................................................................................1
   1.2 Related Work ..................................................................................................2
   1.3 Scope Delimitation ..........................................................................................2
   1.4 Outline .............................................................................................................2

II. METHODS ..............................................................................................................4
   2.1 SOFTWARE DEVELOPMENT PROCESSES ..................................................4
   2.2 DEVELOPMENT TOOLS ............................................................................7
   2.3 PROJECT METHODS ...................................................................................9
       Requirements ....................................................................................................9
       Use Case Description .....................................................................................10
       Spiral development .........................................................................................12

III. DESIGN AND IMPLEMENTATION .....................................................................13
   3.1 DESIGN .........................................................................................................13
       3.1.1 The structure of MOD ........................................................................13
       3.1.2 The process of play MOD files ............................................................15
       3.1.3 The main class of MOD Player ............................................................16
   3.2 IMPLEMENTATION .......................................................................................17
3.1.1 Sampling ................................................................................................. 18
3.1.2 Mixing ....................................................................................................... 19
3.2.3 Buffering ................................................................................................. 19
3.2.4 Playing the sound .................................................................................. 20
3.3 Additional Steps ........................................................................................ 21

IV. RESULT ......................................................................................................... 22

V. CONCLUSION ................................................................................................ 24

VI. DISCUSSION AND FUTURE WORK ............................................................ 26

REFERENCE ..................................................................................................... 27
Table

Table 1: Advantage and Disadvantage between incremental and spiral .............. 7

Figure

Figure 2. 1: The increment model ................................................................. 1
Figure 2. 2: Boehm's spiral mode of the software process .............................. 6
Figure 2. 3: Use Case Diagram ..................................................................... 10
Figure 2. 4: Use Case Scenario: Playing MOD music ................................... 1
Figure 3. 1: The structure of MOD_HEADER ................................................ 14
Figure 3. 2: The structure of MOD_STATE ...................................................... 14
Figure 3. 3: The structure of SAMPLE_HEADER ............................................ 15
Figure 3. 4: The process of playing the data .................................................. 1
Figure 3. 5: Main class of MOD Player ......................................................... 16
Figure 3. 6: the data of note ......................................................................... 18
Figure 4. 1: The menu of Lypson Game Engine .......................................... 23
Figure 4. 2: The mod player of Lypson Game Engine .................................... 23
Listing

Listing 1: The formula of Mixing MOD data........................................................19
Listing 2: The double buffer .............................................................................19
Listing 3: Set the DMA1 channel.......................................................................20
I. Introduction

This chapter describes the origin of this thesis, and the project background, related work, and scope delimitation.

1.1 Background

Game Boy Advanced (GBA) is a handheld game console produced by the Nintendo Company. The sound hardware has a good ability to manage music with complete function, so it can be used as an efficient music player.

In real life, most sound players of the GBA are designed to play wave files or raw files. It is important to remark that the size of these files is measured in megabytes, and consequently, for GBA hardware, the resources of the CPU and memory that are required are too great. Finding a new kind of music file is an inexorable tendency, which is used in GBA sounds system to replace these large files. That is what will be done in this project.

The project described here is proposed by Lypson Intelligent Systems. Since January 2004, the Lypson Company has been developing an API (game engine). The Lypson Game Engine is an object-oriented library, specifically targeted for two-dimensional platform games developed for the Game Boy Advance system. Today, the Lypson Game Engine faces a new problem. It is loaded with "raw" music files, and these music files are too large (they take up megabytes in size). What this API (Game Engine) needs is a new music player, a so-called MOD player, developed specifically for this Game Engine.

What are the MOD files? MOD is a computer file format used primarily to represent music (like wave music files), and the structure of MOD is similar to MIDI which has the real samples. Because of its small size, it is always used for the background music in the DOS. The music file is only a few KBs instead of MBs right now in the MOD player.
The goal of this project is to make the MOD player has the ability to process the MOD music format.

1.2 Related Work

There are many related products which can be found on the Internet. For example, Krawall, developed by Sebastain Kienzl, is an S3M player[^1]. S3M is another music file, which has a small size, but it cannot play MOD music and it is not free for the Lypson Game Engine to use.

The Apex Audio System (AAS), developed by Apex Designs, is a good MOD player[^2], but it has a problem that is not free for commercial use.

There is no good MOD Player which can be used for the Lypson Game Engine, so it is necessary to develop a special MOD player for it.

1.3 Scope Delimitation

In the project, the intention is to develop a MOD Player that plays MOD files. Actually, there are several kinds of MOD files. Firstly, because of time limitation, the player cannot deal with all kinds of the MOD files. Moreover, a design decision was that the MOD Player to be developed would only play the MOD files with four sound channels.

1.4 Outline

Chapter 2(Methods): This section analyses the methods that can be used in the project (development methods), the solutions, and the supporting platforms (like software HAM) used by the project are described.
Chapter 3 (Design and Implementation): The design of the MOD Player is presented in this section, and another important part is the realization of the project.

Chapter 4 (Results): The results obtained from the test of the MOD Player on the company’s system are given.

Chapter 5 (Conclusion): The main findings from the project are summarized and conclusions are drawn from them.

Chapter 6 (Discussion and future work): This chapter discusses more design issues about the MOD Player for future work.
II. Methods

In order to produce a quality project, it is necessary to follow a methodology and to document the processes and results. To perform the current project, some applicable software development process were studied and applied. In this chapter, a theoretical background of these methods is presented, along with the tools used to perform the work. Finally, the process used for the development of this worked is presented.

2.1 Software Development Processes

The software process is a set of activities and associated results that lead to the production of a software product[3]. Two main models about software process presented in summary and compared as follows:

**Incremental model**

Mills proposed the incremental development as a means of the reducing rework in the process [4].

![Figure 2.1: The increment model](image)
As the Figure 2.1 shows, an operational part of software is delivered at each stage of the increment model, instead of the completed software. The software is a series of components, and each of components is analyzed, designed, implemented and tested separately. When a component is completed, it will be delivered to the customer and the customer can put it into service. This allows the customer to review the software during the process of development. This process has a number of advantages:

1. Customers can get a valuable part of software each stage.
2. There is a lower risk of overall project failure, because customers can check the process of the software development.

However, there are some problems with this model.

1. It is hard to map the requirements onto increments of the right size.
2. Identifying common facilities that all increments require are difficult because the requirements of development are not detail until an increment is to be implemented.

**Spiral model**

The spiral model (Figure 2.2) that was proposed by Boehm in 1988 spirals out from near centre. Each cycle of the project needs four steps.

1) Determine objectives, alternatives and constraints: specific objectives for that phase of the project are defined. It identifies constraints, product and risk. The detailed management plan is drawn up.

2) Evaluate alternatives, identify and resolve risks: after the analysis of risks is carried out, it needs to reduce the risk.

3) Develop, verify next-level product: it is based on the last two steps to develop the project. The operation is about requirements, design and testing.
4) Plan the next phase: it is to be decided whether to process with the next loop of the spiral when the project is checked.

The process is a spiral, and each loop in the spiral is a process of development. The innermost loop is the basal process of development.

![Boehm's spiral mode of the software process](image)

**Figure 2.2:** Boehm's spiral mode of the software process

The spiral model indicates that the classic software life cycle model needs only be followed when risks are greatest, and after early system prototyping as a way of reducing these risks, albeit at increased cost.

The spiral model supports the dynamic needs of the users; it is helpful to the adaptability of software. The risk of the project is explicitly considered, and it results in
fewer problems. Nevertheless, it still has some problems. For example, customers need to accept, and believe in, the analysis.

**Compare incremental and spiral:**

<table>
<thead>
<tr>
<th>Model</th>
<th>Advantage</th>
<th>Disadvantage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Incremental</td>
<td>1. Customers can get a valuable part of the software each stage.</td>
<td>1. It is hard to map the requirements onto increments of the right size.</td>
</tr>
<tr>
<td></td>
<td>2. There is a lower risk of overall project failure, because customers can check the process of the software development.</td>
<td>2. Identifying common facilities that all increments require is difficult.</td>
</tr>
<tr>
<td>Spiral</td>
<td>1. Support the dynamic needs of users, it is helpful to the adaptability of the software</td>
<td>1. Developers should analyze risk accurately.</td>
</tr>
<tr>
<td></td>
<td>2. The risk of project is explicitly considered, and it results in fewer problem.</td>
<td>2. Customers need to accept and believe in the analysis</td>
</tr>
</tbody>
</table>

**Table 1:** Advantage and Disadvantage between incremental and spiral

Comparing these two models, each increment should deliver part-functionality. The requirements of incremental model are not defined in detail until the increment is implemented. These are not fit for the project of MOD Player. Rather than the incremental process as a sequence of activities with some backtracking from one activity to another [9], the spiral process is more befitting for the project of MOD Player.

**2.2 Development tools**

**Development Kit:** HAM Development Kit 2.8
HAM is a non-graphical development environment for Game Boy Advance \([10]\). It is useful by itself but it needs to use a graphical front-end, such as Visual HAM. The compiler in HAM is the GNU project’s ARM-compiler. The homepage of HAM is http://www.ngine.de/site/index.php.

**Development Platform: Visual HAM 2.6 Beta 1**

Visual HAM is an integrated development environment (IDE) for the Windows platform \([11]\). By default, it uses the HAM Development Kit for compilation and VisualBoyAdvance for running and debugging the developed games.

**Emulator: VisualBoyAdvance 1.7.1**

VisualBoyAdvance is an emulator for Game Boy Advance that makes it possible to test developed games (as well as commercial games) on a computer. VisualBoyAdvance functions almost exactly as the real hardware. As loading, debugging and error searching is easier on an emulator than on real hardware, the emulator is used for standard testing during development. However, some differences have been observed (i.e. DMA calls are much faster in VisualBoyAdvance) so the game engine must be tested on real hardware too \([12]\).

**Other Software: Doxygen 1.3.7**

Doxygen is a powerful code documentation tool that, among other things, is able to auto generates documentation from C++ files \([13]\).

**Language: C++**

The programming on GBA is an embedded programming, which is solved by three mainstream languages: C language, Assembly language and C++.
It is necessary to point out that C++ can be used for developing powerful applications, and the game engine of Lypson Company is developed by C++.

To make the project powerful and compatible with a game engine, it adopts the C++ as the language for embedded programming.

### 2.3 Project method

This section analyzes the requirements of MOD player in detail. Use cases are used to describe the function requirements of the project, making their understanding clearer. Then it focuses on the process model for the development.

**Requirements**

**Objective:**

The objective of this project is making a MOD Player, which can play MOD music files for the Game Engine of Lypson Company.

**Non-Functional Requirements:**

**Interface:**

A friendly interface should be offered by the MOD Player to the user.

**Performance Requirement:**

- Workability: the MOD player can be put into production.
- Veracity: the result is in step with the operation of user.
- Suitability: the MOD Player is compatible with the Game Engine of Lypson Company.

**Functional Requirements:**

1. The MOD Player is on the list of the computer system menu.
2. The user can operate the MOD Player to play a MOD music, which has been in the Game Engine.
3. MOD music is played in the background.
4. The user can select the music.
5. The MOD Player allows user to stop the music.
6. The MOD Player mixes the 4 channel of Mod music to play MOD music

**Use Case Description**

In order to describe the functional requirements, Use cases for the MOD Player are specified, and they are presented as follow:

![Use Case Diagram](image)

**Figure 2.3: Use Case Diagram**

According the diagram of User Case, there are two actors in this case. One is User, and the other is Administrator.

For User:

1. User chooses the MOD Player from the menu of the Game Engine.
2. After entering the MOD Player, user can press the button to play the MOD music.
3. User can press the button to select the music.
4. The MOD music is playing, and then the user can press the button to stop it.
5. No matter whether the MOD Play is processing or not, the user can exit the MOD Playing

For Administrator:

Administrator can modify the music in the MOD Player. It makes sure the system safety.

**Use Case Scenario: Playing MOD music**

<table>
<thead>
<tr>
<th>Use Case Scenario: Playing MOD music</th>
</tr>
</thead>
<tbody>
<tr>
<td>0. System displays Main Menu (initial state).</td>
</tr>
<tr>
<td>1. User (Actor) selects the Mod Player and it shows the interface of Mod Player.</td>
</tr>
<tr>
<td>2. User presses the buttons to play the mod music or stop mod music.</td>
</tr>
<tr>
<td>3. User presses the buttons to select the music that user likes.</td>
</tr>
<tr>
<td>4. User quits the Mod Player.</td>
</tr>
<tr>
<td>5. System reverts to main menu (final state).</td>
</tr>
</tbody>
</table>

**Figure 2.4:** Use Case Scenario: Playing MOD music

According to the use case, there is a use case scenario about playing MOD music to make the requirement of the user clearer.

When a user opens the system, the main menu is displayed. The user chooses the MOD Player from the menu, and then the interface of the MOD player is shown. The user presses different buttons to control it. When the user finishes the operation of MOD Player, the user can press one button to exit from MOD Player. It reverts to the main menu, which is the final state.
Spiral development

The project adopts the spiral process and the steps of the project are described as:

a) Obtain the requirements by interviewing with Lypson Company.
b) Design a prototype of the MOD Player based on an analysis of the requirements.
c) Get the feedback from the Lypson Company after laying out the prototype.
d) Set the next prototype with regard to the feedback.
e) Repeat the process until the MOD Player is completed.
III. Design and Implementation

This chapter describes the design of the MOD Player at first, and then the implementation follows the design to make a valid MOD Player.

3.1 Design

The first step of the design is the delineation of the structure of the MOD file and the MOD Player. The next step is a process to produce the sound. The last step in the design is to sketch the main class, which will realize the general functionality of the MOD Player.

3.1.1 The structure of MOD

The most important aspect is that the data of the MOD files should be made clear. The format of the MOD file contains 3 components: an order list, patterns, and samples. The order is the number of the pattern to play. The samples contain a small header and raw, 8-bit signed PCM sample data. The header contains properties like the volume, looping points, and tuning. To play a MOD, it should read the entries in the order list and run through each pattern that is listed. The patterns are made up of 64 rows and 4 columns, where 4 is the number of channels that the module contains.
The MOD header and MOD state is shown as Figure 3.1 and Figure 3.2, to load the information of MOD file.

The MOD header is the structure of the MOD file, and it is composed of order; orderCount; pattern and sample.

The state of MOD Player is set two state. One is MOD_STATE_STOP and the other is MOD_STATE_PLAY.

The sample of MOD file records fine-tune; length of wave; loop length; loop start; data of wave and volume.

Therefore, the structure of MOD sample is composed of finetune, length, looplength, loopstart, smpData and vol.

This is shown as the Figure 3.3.
### 3.1.2 The process of play MOD files

1. **Load the data of MOD**
2. **Save the data to the buffer**
3. **Control the hardware of GBA**
4. **Play the data of MOD**

#### Figure 3.3: The structure of SAMPLE_HEADER

- `finetune`
- `length`
- `looplength`
- `loopstart`
- `sampleDate`
- `vol`
For the MOD Player, it should load the data of MOD files first. The detail of loading data will be discussed in section 3.1.1 Sampling. When the data have been captured, they will be saved to the buffer. At the same time, the hardware of GBA will be controlled to send the data directly to the sound channel. This will make the sound of MOD files. These steps are shown as Figure 3.3.

### 3.1.3 The main class of MOD Player

The main class of mod play is as follows:

![Diagram of MOD Player class structure](image)

**Figure 3.5:** Main class of MOD Player
The class of MOD is used to get the data of MOD file. The main methods are these:

“MOD_Play ()”: it is used to set the data pointer in the MOD, which means the MOD Player acquires data by the address.

“MODPlayNote ()”: “MODProcessRow ()” call it when it should set the value to the members in the sound channel.

“MODProcessRow ()”: this will update the data of the current row.

“MODUpdate ()”: when the tick gets to the speed, which means the row needs to be updated, “MODUpdate ()” will be called.

The class of sound_vars uses double buffer to store the data. This makes the MOD Player more efficient. There is a method (“SndVSync ()”) swapping the double buffer by the VB interrupt.

The class of MOD_Player is used to control the hardware of GBA to play the data of MOD. There are two methods to which attention should be paid. First is “void start()”, which initializes the state of the MOD Player. The other is “void play()”, which is used to play songs.

### 3.2 Implementation

The process of implementation follows from the design process. The MOD Player is implemented on Windows XP, and C++ language is chosen. The implementation has five parts to achieve the project. The first step is sampling which is capturing the data of MOD music, and converting the data to a binary file for GBA reading data. When the MOD Player takes samples, it mixes these data to prevent them from distortion. After mixing, the data can be played. It needs to point out that it will create a buffer system for
MOD Player of GBA, then the MOD Player is efficient. When the preliminary test of MOD Player has been passed, it can be embedded into the Lypson Game Engine.

3.1.1 Sampling

This step converts the MOD file to a binary file\[16\].

1) If the four bytes located at 1080 (438h) in the file are "M.K.", it is a four channel that is needed.
2) The 20 bytes at the start of the file has the name MOD. Define it as char modname.
3) Sample: there are 31 samples.
   For each sample: 22 bytes, sample name
      2 bytes, sample length
      1 byte, fine-tune
      1 byte, volume
      2 byte, loop start
      2 byte, loop length
4) Load order: order defines in what order patterns are going to be played
5) Loop 128 times to get the number of patterns; after loop, read 4 bytes "M.K."
6) Load pattern data: The size of one pattern is 4*4*64 bytes.
   Note of pattern: It is 4 bytes. The structure of the data stored in note is shown as:

   Listing 3.6: the data of note \[17\]

7) Load Sample Data:
   This loops 31 times to get the length of sample, and then acquires the wave data.
   Those wave data are stored in the mod_Data.cpp.
3.1.2 Mixing

The first step of playing a MOD music on GBA is mixing. The mixing code [18] is shown as:

```c
for(curChn=curChn<4;curChn++){New data=data[data.position]<<12]*vol>>6;}
```

Listing 1: The formula of Mixing MOD data

The channel of MOD is four, so the time of the loop is four. New data obtains the data of current position, when the data have shifted 12 positions. The six is volume levels, in which the music format uses 64 levels.

Next step is that the mixed data are sent to the intermediate buffer.

3.2.3 Buffering

For the GBA, a buffer system is needed to load the sound data, and a double buffer is the best choice.

First, set two buffers with the same size, one is "mixBufferBase"; the other is "mixBufferBase + mixBufferSize". By the way, there is a regular for the buffer size: The frequency of the buffer is 18157 hertz. The timer for swapping is (655536- (167772116-18157)) =64612, and the size of buffer is 304 bytes [19].

```
if(activeBuffer==1){curMixBuffer=mixBufferBase+mixBufferSize;activeBuffer=0;}
else {curMixBuffer = mixBufferBase;activeBuffer = 1;}
```

Listing 2: The double buffer
When the data have been mixed and saved in the "tempbuffer", the "tempbuffer" will send it to the current buffer. When the current buffer is full, it will generate an interrupt to swap the current buffer for another buffer.

3.2.4 Playing the sound

The work of the data is finished, and the rest is playing the data. To control GBA, play the data; this should set registers of hardware.

One is DMA (DMA is a special line used for transmitting data and it can transfer data without a CPU). There are four DMA channels. The DMA transmits the data to direct sound.

Direct Sound refers to the two, 8-bit digital-to-analog converters part of the Gameboy Advance sound system (GBAS). The samples to be played, which must be 8-bit signed, are loaded in consecutive addresses, starting at 0x40000A0 (REG_FIFO_A). These addresses act as a FIFO (First-In-First-Out), meaning that lower addresses bytes are played first.\[20\]

First, it sets the source address of the DMA to the sample address and destination address to the FIFO address. FIFO is set before starting the sound by setting the FIFO reset bit, and the DMA is also be set.(Listing 3 lists what set the DMA1 channel).

\[
\begin{align*}
\text{REG\_DM1CNT} & = 0; \\
\text{REG\_DM1AD} & = (u32)\text{mixBufferBase}; \\
\text{REG\_DM1CNT\_H} & = \text{DMA\_DEST\_FIXED | DMA\_REPBAT | DMA\_WORD | DMA\_MODE\_FIFO | DMA\_ENABLE};
\end{align*}
\]

**Listing 3:** Set the DMA1 channel

This project uses DMA1 to connect the source address with Direct Sound A.
3.3 Additional Steps

1. The last step of the project is to combine the MOD Player with the Lypson Game Engine. The most important was the modification of the code to prevent conflict. For example, the Lypson Game Engine has turned off the switch of sound, and then the MOD Player should turn it on first to play a MOD file.

2. The instruction of MOD Player: Press button “START”, and then MOD Player start playing MOD music. Press button R, and then the MOD Player plays next song. Press button L, and then MOD Player plays previous song. Press button A, and then MOD Player stops playing. Press button B, and then it exits from “MODplayer”.

3. See Appendix for details about GBA register.
IV. Result

This section covers the validation of the MOD Player which has been designed, and its features are explained. It summarizes the result obtained by testing.

The MOD Player is tested on Windows XP, and the emulator is VisualBoyAdvance, which makes it possible to test the development of GBA on a computer.

Testing was divided into two parts. The first testing was of the MOD Player before it was embedded into the Game Engine. The major issue here was that the size of the Game Engine is large, and it takes more time to makefiles.

The other testing follows the work of embedding the MOD Player into the Game Engine, which was the last step. This ensures the MOD Player is usable by the Lypson Game Engine.

The first testing:

1. Convert a MOD file
   The tool of convert is “converter.exe”, and the command “converter data” calls the tool to convert MOD files. The convert was successful, and then it generated a file (SndData.cpp) to save the data.

2. Send data to the MOD Player:
   The “SndData.cpp” was added to the system. It is clear that the data is saved to the buffer when the program is debugged in visual HAM. The visual HAM provided a GBA file which is generated by making file.

3. Run “MOD.gba” on the emulator
   The GBA file of project was called “MOD.gba”. When the “MOD.gba” was run on the emulator, the sound was produced successfully, which means that the MOD files can be played by the MOD Player.
**Second testing:**

The MOD Player has been embedded into the Lypson Game Engine. When the Game Engine is enabled, the menu arises. The menu is as follows: Play demo; View cinema; View credits; Mini games and MOD Player.

A cursor, which is controlled by the “up” and “down” buttons, points at the selected command.

![Figure 4.1: The menu of Lypson Game Engine](image1)

As shown in Figure 4.1, the cursor points at “Mod Player”. When press the button “start”. Then MOD Player is chosen from the menu.

![Figure 4.2: The mod player of Lypson Game Engine](image2)

As Figure 4.2 shows, it proves that the interface of MOD Player is on the right way. Upon pressing the “start” button, the MOD is playing music. The buttons accord with the operation. The test is finished.
V. Conclusion

The project is based on the GBA, so it was necessary to gain knowledge about how to develop embedded software in this platform and the structure of mod music.

Through the development of this project, it was possible to get the necessary knowledge about the hardware of GBA and we managed a good command of programming based on GBA. The language of project requirement is C++, so it was also a great opportunity to improve our skills in C++ programming. In addition, the project also gave us a chance to learn about embedded system in general. There are still some problems in the development which is worthy of mentioning, but which points to directions of possible future works.

The problems of project

The main problems that interfere with the development of the process are these.

1. Convert the mod file:

   The mod file should be converted to binary file first. It is hard to find an efficient tool to do it. A tool that can capture the data of mod file is chosen, and the tool is modified to generate a CPP file that includes the data of the mod file.

2. The same address is occupied by two functions.

   This is a programming problem. There are some data that are changed by another function, and then find the same address is occupied by two functions. The method is constructing these functions to give them an address.
3. Mod cannot play a big mod

This arose on the process of testing. The MODplayer cannot generate the sound with the large MOD file. We redefine the SndData.cpp and add these:

```cpp
#define EWPARAM_DATA __attribute__((section(".ewram")))
#define EWPARAM_CODE __attribute__((section(".ewram"), long_call))
```

Listing 5.1: Definitions added to SndData.cpp

4. Interrupt

The Game Engine has an interrupt function and when this is used, the MODplayer cannot work. It takes a long time to check. Lastly, we find the function of the Game Engine needs a boolean value to use the interrupt, and it is different from our interrupt.
VI. Discussion and future work

Nowadays, there is not an efficient MOD player on GBA, and the ones that are available, like Krawall or Maxmod, are not tailored for Lypson Game Engine. In this document, a MOD Player for the Lypson Game Engine has been presented. It solves the problem that the sound system of the Lypson Game Engine just plays wave files, and it takes up much of the memory resources. In the scope of the thesis, the MOD Player is completed successfully. The MOD music can be played in the Lypson Game Engine.

However, we still have much future work to do. The MOD Player can only play the four channels MOD files. Therefore, it should find a solution to play the other small music files, such as s3m (by the way, s3m can be played by krawall, but krawall is not for commerce).

Moreover, the effect needs to be combined with a MOD wave sample. About this, we have collected effect information and some ideas have been in consideration.
Reference

[16]. MOD Player Tutorial by FireLight,


[18]. Sound on the Gameboy Advance, URL: http://deku.rydia.net/program/sound2.html, 2010-05-19


Appendix A

Set the register of GBA:

#define REG_SGFIFOA  *(volatile u32*)0x40000A0
#define REG_SGFIFOA_L  *(volatile u16*)0x40000A0
#define REG_SGFIFOA_H  *(volatile u16*)0x40000A2
#define REG_DM1SAD  *(volatile u32*)0x40000BC
#define REG_DM1SAD_L  *(volatile u16*)0x40000BC
#define REG_DM1SAD_H  *(volatile u16*)0x40000BE
#define REG_DM1DAD  *(volatile u32*)0x40000C0
#define REG_DM1DAD_L  *(volatile u16*)0x40000C0
#define REG_DM1DAD_H  *(volatile u16*)0x40000C2
#define REG_DM1CNT  *(volatile u32*)0x40000C4
#define REG_DM1CNT_L  *(volatile u16*)0x40000C4
#define REG_DM1CNT_H  *(volatile u16*)0x40000C6
Appendix B

CLASS:

Sound Channel:

class sound_channel
{
    protected:
        s8    *data;
        u32   pos;
        u32   inc;
        u32   vol;
        u32   length;
        u32   looplength;

    public:
        sound_channel();
        void channel_init();
        void channel_mix(s16 *buffer, u32 samplesToMix);
        void set(s8 *d, u32 p, u32 i, u32 l, u32 loopl);
        void set_vol(u32 volume);
        void clear();
};
Mod Channel:

class mod_channel {
    private:
        u32 frequency;
        u8 sample;
        u8 vol;

    public:
        mod_channel();
        u8 get_sample();
        u8 get_vol();
        u32 get_frequency();
        void set_sample(u8 sam);
        void set_frequency(u16 freq);
        void set_vol(u8 volume);
};
Mod:

class mod {
    protected:
        sample_header *sample;
        u8 **pattern;
        const u8 *order;
        const u8 *rowPtr;
        u8 curRow;
        u8 orderCount;
        u8 curOrder;
        u8 state;
        u8 speed;
        u8 tick;
        u8 tempo;
        mod_channel channel[SND_MAX_CHANNELS];

    public:
        mod();
        u8 get_state();
        void MOD_Play(mod_header *m);
        u32 MODUpdate();
        void MODPlayNote(u32 cldx, u32 note, u32 sldx, u32 effect, u32 pattern, u16 rMF, sound_channel *sc);
        void MODProcessRow(sound_channel *sc, u16 rcpMixFreq);
    };
Sound Vars:

class sound_vars
{
    protected:
        s8        sndMixBuffer[736*2];
        static s8  *mixBufferBase;
        static s8  *curMixBuffer;
        static u32  mixBufferSize;
        u16       mixFreq;
        u16       rcpMixFreq;
        s16       UntilMODTick;
        u16       PerMODTick;
        static u8  activeBuffer;

    public:
        sound_vars();
        static bool SndVSync();
        void     vars_init(SND_FREQ freq);
        void     vars_mix(s16 *buffer, u32 samplesToMix);
        void     vars_Tempo(u32 tem);
        u16      get_rcpMixFreq();
};
class mod_player: public mod, public sound_vars
{
    private:
        s16 tempBuffer[304];
        sound_channel SndChannel[4];

    public:
        mod_player();
        void MOD_Init(SND_FREQ freq);
        void MODSetTempo(u32 tem);
        bool SndUpdate();
        void SndMix(u32 samToMix);
        void SndPlayMOD(u32 modIdx);
        void start();
        void stop();
        void play();
        void disable();
};