**Analysis**

Fully or nearly fully scoped analysis of a real problem, presented in a way that a third party can understand.

Requirements fully documented in a set of measurable and appropriate specific objectives, covering all required functionality of the solution or areas of investigation.

Requirements arrived at by considering, through dialogue, the needs of the intended users of the system, or recipients of the outcomes for investigative projects.

Problem sufficiently well modelled to be of use in subsequent stages.

Think of product name

Put table in design of functions and their purposes

**Background to Problem**

The MIDI file format is a widespread and established standard for storing music. The music macro language (MML) is a niche language for describing music in a comparable way to how music is stored in MIDI files. It has no real standards, but has evolved over the years into different versions, the use of which are provided by BASIC implementations. One of the key issues with the MML is how there is not much software available that can play it; another issue is that the MML is, by design, only suited for monophonic tracks.

This project seeks to tackle both of these issues by providing a facility that can convert a number of MML files into a single multi-channel MIDI file. This allows the benefits of both the MML and MIDI file format to be taken advantage of – music is easily entered with MML, and it is easily playable using the MIDI format.

**Research Methods**

I will use the MIDI specification document supplied by the MIDI association to learn how the MIDI file format is structured.

The best resource I have found to research the music macro language is the Wikipedia page.

Section exact workings of MIDI, with examples

Makefiles

Lex

Yacc

MML mention

Endianness

**Description of the Current System**

The current way of converting multiple music macro language like files into a single multi-channel MIDI file is to step through by hand and enter the information as if writing a new MIDI file. This is tedious, unreliable and slow.

**Identification of Potential Users**

There is a small, but dedicated, remaining group of musicians that still use the music macro language

**Identification of User Need and Limitations**

Users will need to be able to use all of the syntax of the MML, as it would be difficult to enter music if even one feature was removed.

The users should not need to have any technical knowledge to use the solution.

The solution should be fast enough to keep up with a musicians work flow, otherwise this would be stifling for their creativity.

The syntax of the MML used by the solution should be at least similar to what already exists, so as to minimise the effort required by the user.

Fake user interaction, meant to be a product

**Data Flow Diagrams**

The following flow chart shows how the solution to the problem should work in terms of file flow:

**F:\School work\A Level\Computer Science\MML-To-Midi-Project\Project Documentation\Diagrams\Analysis_Data_Flow.png**

**Use Cases**

A musician whom uses the MML to compose music would likely find the tools written for this project very useful, as it allows the conversion of their preferred format to an easily playable and portable form.

The programs would also very suitable for a less musically experienced individual, as it is easy to alter and enter music in MML, in contrast to traditional sheet music.

**Potential Solutions**

One solution to the problem is to have one terminal program that the user enters the paths to the input MML files into, and have it output the desired MIDI file. There are a number of pros and cons in this potential solution:

|  |  |
| --- | --- |
| Single Program Advantages | Single Program Disadvantages |
| Easy to use | The program would be complex, given what functionality it must have |

Another solution would be to have two programs that are designed to be used in conjunction: the first would take a single MML file and output a single-channel MIDI file, and the second would take multiple single-channel MIDI files and output a single multiple-channel MIDI file. Some advantages and disadvantages to this approach are shown below:

|  |  |
| --- | --- |
| Program Pair Advantages | Program Pair Disadvantages |
| Each program would be more simple to write, compared to a single complex program | More difficult for to use |
| Each channel of the final MIDI file can be listened to, as a single-channel MIDI file of it would be generated before-hand |  |

**Chosen Solution**

I have chosen to use the solution featuring a pair of programs. This is primarily because I feel that the complexity of both programs combined would be less than the complexity of one larger program.

**Other Evidence**

Research that doesn’t fit into any other categories.

**Objectives of the Project**

* A program should be written that takes text file containing a variant of the MML as an input, and outputs a single-channel MIDI file that can be play with conventional software.
* A program should be written that takes multiple MIDI files generated by the aforementioned program and combines them into a single multi-channel MIDI file. This combined MIDI file should be playable using conventional software also. – follows file structure and format
* The programs written for this project should:
  + Complete their execution in under one second, so as to not interrupt the users work flow
  + Use a reasonable amount of memory during execution – make specific
  + Be a reasonable size – make specific
* A version of the music macro language should be designed that will be used as the input for the program that generates a single-channel MIDI file from a single MML file. This language should:
  + Have all the functionality of existing variants of the MML, including support for:
    - Octave changing
    - Accidentals
    - Default length setting
    - Volume changing
    - Tempo setting
    - Macros
  + Have as unambiguous a syntax as possible, with a clear logical progression

Bibliography