Lights Program
Software Requirements Specification
For Programmable Animal

Version 2.0

Revision History

Date	Version	Description	Author
early 2014	1.0	Original version. Was only able to handle single keyboard input and required 12 handwritten files per song. Ran on Linux	Eric Lazarski
late 2014	1.5	This version was never finished, it was the first attempt at handling multiple keyboard inputs and reading from standard files. Ran on Linux	Eric Lazarski
early 2015	1.7	First version to not use C/C++. Again, this version was never finished. It ran on Python. Last version to run on Linux	Eric Lazarski

Table of Contents

Software Requirements Specification

1. Introduction

This requirements specification discusses why the Lights Program is being developed. It explains the scope of the project, specific use cases for it, as well as what was assumed during the design process.

1.1 Purpose

This specification is written so that the Lights Program can be fully understood and planned out.

1.2 Scope

The Lights Program is designed to send signals to a DMX program which will, in turn, send signals to DMX lights. This will give us both a personalized and synchronized visual show. The setup that we will have is a computer for our keyboardist as well as DMX lights and the necessary cabling for both.

1.3 Definitions, Acronyms and Abbreviations

DMX stands for Digital Multiplex. It refers to a standard method of communicating with lighting systems. It used either a 5-pin or a 3-pin connection. It can also be sent through Ethernet.

API stands for Application Program Interface. It allows a developer to access features that are not included by default in a language.

MIDI is an acronym for Musical Instrument Digital Interface. It is a protocol that was developed in 1979 for synthesizers. The protocol has not changed as it has been extensively used ever since its invention.

A programming library is a set of compiled code that gives the developer access to a feature. It can be included in the language itself or be a part of an API.

I/O is input and output. It refers to reading from or writing to a device or file.

1.4 Overview

The rest of this document explains assumptions that were made during the design process, and specific requirements for the project. It also explains the reasoning for the project's development in the first place.

2. Overall Description

The program was designed with previous shows in mind. There have been many instances where the lighting engineer is also the sound engineer so they will leave the lighting system alone during our set. This program is being developed so that we are able to create personalized light sequences that are also synchronized with out music. Functionally, it will take input from our keyboardist who uses MIDI keyboards, parse the notes it receives, and, when the time comes, it will send a signal to a DMX program.

2.1 Use-Case Model Survey

The main use case for this software is taking input from our keyboardist and parsing it. Other cases prior to that are selecting songs or a setlist to play and selecting MIDI devices to be used.

2.2 Assumptions and Dependencies

I assumed that the languages available to me would have API's to interface with MIDI devices and possibly DMX devices as well. I also assumed that they would have libraries to access certain file types.

The project is dependent on being able to interact with MIDI devices, but not on reading files. It is possible to read what is necessary in a bytewise manner and parse them that way. However, it is dependent on being able to run in real time. The software needs to be as quick as possible with the language that is being used.

3. Specific Requirements

As mentioned above, the Lights Program needs to run in real time. If it does not, it will be entirely pointless as the signal to the DMX program will not be sent on time. It also requires that MIDI I/O be handled without error. It is required to handle error from our keyboardist while staying on time with what is being played.

3.1 Use-Case Reports

On error handling, the software must be able to handle it when our keyboardist misses a note and hits another nearby. It also needs to be able to handle when our keyboardist entirely misses a small section of a song because he got out of sync with the rest of the band. This could mean he skips forwards in a song or goes backwards. If we cut a part too early or keep it going for too long the software needs to stay with the band rather than continuing with what it read from the file. On the subject of files, it needs to be able to read some sort of standard file for musical notation. This way, the files can merely be exported from GuitarPro or TuxGuitar into one of these formats and then read from the Lights Program. It also must have as few errors as possible, as it will be used on stage where quick setup times are a must.