Great background. This help me better understand what you are planning to do. Nice reflection on outcomes.

Project Proposal – Lights Program v2
Eric Lazarski

Introduction:

My project is to finally finish the Lights Program for my band. I have been working on this for about two years now, and this will be version 2 of the software. I am writing it so that we will be able to have a live visual show without having to play to a 'click' (a pre – rendered track that we have to stay on time with) or paying someone to control the lights system. Even if they were unpaid, having another person with us would put six people together, when five was already overly difficult enough to schedule and work together as it was. We used to be a six piece band, and it was extremely difficult to schedule shows and practice times. This method becomes easier in the long run, as we will be able to do more with the program than a human would be able to do with a DMX board. DMX is a standard for stage lighting, similar to TCP and Ethernet.

The main reason that I decided to write this program is because of my personal experience at shows. Our own shows, as well as others', factor into this. A venue will have a sound engineer also working with the DMX lighting board. They will almost always default to sound and will not touch the second board unless necessary once they have found a set of settings that they like. When audio and visuals are mixed properly, a band's performance goes from good to spectacular. When they are mixed poorly, the performance can degrade. This has happened to us countless times; we will have a slow song, and the person who is controlling the lights has the strobes on full blast. Admittedly, they are almost always also running sound, so I understand why lights are neglected. However, I wish to change that situation, at least for us.

With this program, we will be able to explicitly define how our lighting system responds to our music. With this program coupled with an Android App I wrote a few semesters ago, we will also be able to give queues to the person running the lights at a venue, so that they are aware of what is coming and can adjust the system accordingly. If we are lucky and the venue allows us to control their lighting system, we will be capable of controlling their system so that it is perfectly synchronized with our playing. After everything has been implemented and set up, I would like to think that our stage presence will go from average to phenomenal.

In this proposal I will explain how the program and system that I will have works in general, how I am going to implement version 2, why a reimplementation is necessary, and what will be included in this newer version.

Statement of Work:

The software operates by taking input from MIDI keyboards, which our keyboardist already plays, and based upon the input it receives it will be able to keep track of our position in a specified song. Based upon this position, it will send out signals to another program, called Q Light Controller+, a DMX lighting program, which will in turn send signals to our on stage lighting system. If a venue allows, it is also capable of controlling their system as long as I have configured it properly to do so. This is unlikely, however, so I wrote an Android App that will give queues to the person who is controlling the system. The queues will not be particularly complicated, just enough so that they know what they should not have running, perhaps color requests so that the mood of a song is easily understood.

Version 2 of the Lights Program is being written in Java. This is not by choice as much as it is out of necessity. Windows does not have built in MIDI routing, so I had to download software that creates virtual MIDI thru ports. I tested it with our synthesizer software and it was able to connect. My next task was to find a programming language that would also connect to these virtual ports. I tried

C/C++, but the documentation on the Windows MIDI API was nearly nonexistent. With the little I found, however, I was able to write a short program that would list available connections. The ports I created were not available. Python was next. It had no MIDI libraries by default, so I had to download them and configure them. Again, the ports were not visible. Ruby, once the required libraries were installed, saw the ports, but did not parse input from them correctly. Java was the only language that was able to connect and reliably receive input. None of the above languages were capable of creating a virtual MIDI port because this is unsupported in a Windows environment without hacks or multiple additions.

Version 1 was able to handle input from a single keyboard, it had no UI other than a command line argument that specified what folder to load files from, and required at least 10 hand written files per song. I could have written a program to expedite the writing process by automating nearly all of it, but then I decided that it would be better to just reimplement it with more features. Version two has a full GUI, it requires a single file per song and can also load setlists, which are multiple songs in a predefined order. It also can handle input from up to 16 keyboards. In order to stay operational in real time, I will be using concurrency principals, such as threading and process creation. As of my current plan, each input will have a thread of it's own, as well as each output. Java can handle input from that many sources in real time in a single thread. The reason that I am giving each input a separate thread is because of the extensive error handling that I want to implement. This is discussed below. The main program loop and the GUI also have separate threads running. The final thread will be a TCP server that allows the app to connect. Again, this is discussed in more detail below.

Another reason that I am rewriting the program is because of error handling. Version 1 was only capable of handling user input that was a single step off from what was supposed to be played. I want to implement version 2 so that it will be able to handle octave jumps, temporary improvisation, and changing parts of a song earlier or later than expected. It will handle error by keeping track of often-played melodies and chord progressions, checking if we are playing one of those rather than where the variables point it to. Also, it will accept one more input on top of the 16 mentioned above. This will be another MIDI keyboard, but this one will be directly connected to the software, rather than going through our synthesizer. It will be a control keyboard. Predefined keys will start and stop a song, move to the next part or repetition. It will also allow us to manually move backwards as well or start and end an improvisation section.

errors

Currently, I have almost finished the GUI. I am currently writing code to connect to MIDI devices based upon user selection. After this, I will begin the major part of the back end; MIDI input processing and output when required. Luckily, I already have the majority of the logic written elsewhere so it is merely a matter of translating languages and working out bugs that I was unaware of during my initial development.

In the end, I will also implement a TCP server that will allow an Android phone to connect when running the Lights App that I implemented a few semesters ago. This will send a Java char or int to the phone over the network and the phone will display a text message to the sound/lights engineer at a venue to queue them. It will also, eventually, allow downloading of new files for songs on the app. There will be an editable configuration file that describes a directory for the server to push files from. The app will request a file listing and then the user will request a file to be downloaded. The server will push this file to the device, which will parse it, making sure that it is properly formatted, and then put it with the other files to be used during a performance. It will be able to download single songs and setlists as well, meaning a list of songs to be played, so that it will be a similar experience on both of the two applications. In order for a setlist to work, all of the songs it uses must be on the device, of course.

If I can manage all of this, I will also reimplement the app so that it will interface with this

This sentence is an “orphan”. Try to adjust spacing or add page breaks to avoid this.

version of the Lights Program. Currently, it only works when it received MIDI data from a Linux machine through ALSA. As far as the app goes, I will redesign the UI so that it is better structured and more user friendly. In the back end I will be able to remove the C library that I had to implement and will be able to use pure Java. The second version of the app will be considerably more simple than the original when it comes to program design, as I will understand what has to be done in order for it to work. This means that it will have a more standardized code style, rather than multiple styles that come from researching how to perform specific actions and using the code found on the Internet.

Outcomes:

When I have finished this version of the program, I will have learned about concurrency in Java. This is something I have not really dealt with before, so it will be extremely interesting. I have touched processes in Python and threading in C++ on Linux, but that is the extent of my experience. Once I had the basics of programming down, I was extremely interested in how to make my programs run multiple tasks at the same time without causing one to stop. I am looking forward to understanding the intricate details on this subject.

Secondly, I now have knowledge about GUI design in Java. Even if I am merely using a WYSIWYG editor, I am still gaining at least a small amount of knowledge from having to edit and add code to what is automatically generated. I used to be adept at this in Visual Basic, as my first experience programming was with that language in high school. I have not touched it since, though, so it is not my strongest point. GUI, in general, is one of my weakest points when it comes to developing an application. The only UI design that I have done since then was in Android apps. A GUI in Java in a windowed form is not something I have ever done before. Personally, it is not of much interest to me as I am more intrigued by backed processing. It is a valuable skill, nonetheless, so I am not going to dismiss it as if it were useless.

The Java MIDI subsystem is another set of skills that I will be able to be fully aware of after the completion of this project. It is not as easy to understand as ALSA on Linux, which basically had an input pointer that could be polled and an output pointer that would be flushed. In Java, on the other hand, there are transmitters and receiver objects that are used to parse input. A transmitter object refers to output and a receiver refers to input. The complicated part is because a transmitter object has to be created for each input so that it can transmit to a receiver. The receiver is the object that parses the MIDI signal, rather than being available to the main program method directly. Also, my program has to connect on its own. In my previous version with ALSA, I could rely on a third program to do the connecting, now I will have to fill that role as well. As mentioned above, this is the current section that I am working on.

Finally, I will have much better debugging and coding skills than before, as I know that the playing back end is going to give me an extremely difficult time, especially with the extra error handling that was not present in the original version. In my previous versions, this is what took me the longest to get right. I have also never done it in a concurrent fashion, always a single thread with a single input. I started concurrent inputs in an unfinished version, but did not get far. I soon abandoned that version because after months of writing a library to read a filetype, MusicXML, I realized that it was still not parsing them quite correctly. Specifically, I was unable to get it to read the timings properly. That was also the last implementation in C++. Hopefully this will easier than before, as Java has more debugging abilities than C++ does, especially because my IDE, Eclipse, was originally designed for Java development.