Eric Lazarski
Software Systems Capstone
Reflection Paper

I started this project approximately 2 or 3 years ago. It was my first summer working at Lewis and we student workers had little to do. I decided that I would write a program that would somehow control lights (and possibly more) on stage for my band while we played. I originally had no clue where to even start. I now know, looking back, so much more than I did as a student fresh to Lewis. This document shall explain the journey that I have taken throughout the development of this project until this date.

As mentioned above, I started this project with only a vague idea of what I wanted it to do and how it would work. At that time, I did not even have a clue as to how to actually control a stage light with a computer. I did some research and I discovered DMX, a standardized protocol for lighting systems. I also found that DMX programs exist on various platforms. I looked into some and found that most were rather expensive, so that would not be particularly viable. Continuing my research, however, I found one that was promising. It was called QLC and ran on all three major operating systems. After watching a quick tutorial, I discovered that I could control it with MIDI signals. I now had an idea of how my project would function.

The project now had a direction; to take MIDI input, parse it to know where the musician is in a song, and then output MIDI notes to QLC+ (an updated version of QLC) which would in turn control a sequence of stage lights. I started to code. My first language of choice at this point was C++, and the only reason I chose it was because I had heard that it was incredibly fast compared to the other languages I knew of. And thus I began learning C++ while writing it. This initial version is written for Linux, as it was my original intent to automate the entire process. All our keyboardist would have to do is turn the computer on as I would have preconfigured it for the show before we played. This took me about a year to write if I remember correctly. I learned about the MIDI protocol, how to code a basic application in C++, and about the Linux audio system, ALSA. I actually got this version to work, however, the code was messy and probably had a terrible memory leak. Also, it could only handle a single keyboard, it was reactive rather than proactive, and our keyboardist had to play the song perfectly in order for it to work. This was not feasible for a live performance setting, and so I began development of my second version.

The second version of my lights program was going to be a huge improvement upon my first. It was going to read in a standard filetype that could be generated, it was going to handle a variable number of keyboards for input (and output), and it was going to be proactive rather than reactive by

handling timing. This version also was designed for Linux. My first goal was to read in a standard filetype rather than 12 I had to manually generate. I started testing different filetypes. I could not read a MIDI file in C++, or at least I could not find a free library to do so in a way that was beneficial to me. Lilypond was equally as useless, if not more so because I think the plugin that exported them was broken. I settled on MusicXML in the end as that appeared to be okay and I could understand it. MusicXML, I soon discovered, has a massive schema, and there were very few C++ libraries that would read it. I could not find one that was free that would do what I needed it to, so I set out to write my own library to do so. After about 6 months or so I finally decided that I was reading the XML files correctly and I moved on to threading the input parsing. I soon discovered that I was wrong about being able to read the files correctly. In this version I learned about the MusicXML schema and a little on how to thread in C++ on Linux, but I scrapped the version at this point. I tried to do the same thing in Python. There was a library that read in the files correctly and I was able to use separate processes for parsing input and output. After I got this far I decided that it would be best to just have a single computer on stage, so I attempted to port my code over to a Windows environment.

There is no fully functional MIDI API in Windows. Therefore, I had to download a simple program to create virtual MIDI ports that applications could connect to. There was a problem, though. Python was unable to recognize these ports and would only connect to physical MIDI devices. After this, I tried using C++ again, but on Windows this time. After many headaches, I discovered that the limited API was also unable to see the ports. My next choice was Ruby. After downloading a library that would supposedly allow me to parse live MIDI data, I discovered that it was able to read data from the virtual ports I had created. I started legitimately coding again. I soon discovered, however, that upon receiving a single note from one of these ports, I would get random data and the notes that were being played to the port was not being recognized. Finally, I looked into Java. This language is the only one I found that properly connects to and interfaces with these virtual MIDI ports. This is where the current version of the Lights Program began. While beginning to write for Windows, I discovered the complete lack of documentation available for developers and questioned how other people are able to write software for it. I also discovered how difficult it is to just start to code on the operating system. In order to compile a C++ example, I needed a developer license. In order to open the example, I needed an ancient version of Visual Studio. I had to deal with path issues when setting up Python and Ruby. All in all, it was a terrible experience to set my Windows installation up for development.

Eric Lazarski
Software Systems Capstone
Reflection Paper

Finally, I started this final version of the Lights Program. Written in Java, I created a GUI for the first time in years. I learned how to create a window and have button clicks call functions. In order to start that window, however, I needed a separate thread for it. I started asynchronous programming very early in this version. I learned how to start a thread after creating it, and I also learned how to communicate with it one it has been started. One of the major difficulties at this point was staying in real time without spiking the CPU. I discovered that queues have a *poll* method, so I started using that with a minuscule timeout. After I eventually got the UI functional enough, I started working on reading files. Java has a built in library that can read MIDI files, so I used that originally, but I eventually discovered that I was unable to extract the data I wanted with this library. I tried a few other filetypes like before, but eventually settled on writing a Ruby script that would convert a set of MIDI files to a set of easily readable files in Java. I then learned how to interact with MIDI devices in Java. Luckily this was simple enough, so I was able to quickly get started on the playing backend of the program.

I discovered that this would be much more difficult that I thought it would be. Moving through a song when it was perfect was easy, and I was able to implement that in about an hour. I was able to handle simple error handling (hitting the wrong key or just playing nonsense) in another day or two. However, I ended up being unable to properly handle error where we skip around in a song compared to what the program expects. In order to do this, I will need to reimplement how I am storing the song in memory. I'll need to use a more object oriented approach to the song and its parts. So I moved ahead with making the system proactive with its output rather than reactive like its predecessors. I added enough lines to create a clock after fixing logical issues. These had me stuck for almost a week, but I managed to fix them before the presentation. I will need to do some cleaning here for the final version of the program. As of right now, it still has issues with timing, but it gracefully handles when the song is being played faster than expected. It does not do as well when the song is being played slower than expected.

These past 3 years have been quite a journey with this program, and I am truly glad to see it nearly ready to be used. I hope that I can finish it very soon because we are intending on going on tour in a few moths, so it would be incredibly beneficial to have it ready.