Project Postmortem Report

Object-Oriented Design and Analysis

CS 3307

December 8th, 2021

Group 14

Joseph Siy

Brian Cheng

Patrick Mihalcea

Usman Khan

Nada Elkelani

1.0 Project Summary

Traffic Light Lamp is a ‘smart’ party lamp that changes colours, brightness, and light patterns based on user input. Our interactive traffic light has an option of changing between three colours, 255 brightness levels, and four different patterns. The controller of this project is accessible to all, with the ability to read out what options have been chosen by the user. This project has both a software and hardware component.

1.1 Hardware Component

The hardware materials used as part of our project are as follows:

- Raspberry Pi 3 Kit
- WS2812B LED light string
- 5V Power Supply
- Logical Level Shifter (3.5V to 5V)
- Television
- HDMI to HDMI cord

The breadboard from the Raspberry Pi kit was where all the hardware was attached. The 5-volt power supply was necessary to power the LEDs. The Raspberry Pi has some power output, but not enough for the full strip of LEDs at maximum brightness. More was needed, hence we purchased a power supply. The logical level shifter was used to convert the 3.5-volt signal coming from GPIO 18 of the Pi to a 5-volt signal for the LED lights.

1.2 Software Component

The software component of this project is twofold.

(i) A controller used by the user to select different light options has been created using Qt 5 written in c++ code.

1.2.1 Controller Pre-set Options

A user can pick up to three settings and save them into one of the state buttons we created in our controller. The three settings can save up to two different options, one set for the brightness level and the other set for the mode based on the patterns we developed for our project. As a user picks their settings, the options will appear in the output box at the top of the controller. This way, the user can know exactly what options they have chosen and have saved into what state after reloading the settings. If a user decides that they like the combination they have chosen, they are able to save these settings in one of the state buttons. This is done by pressing the save button, then pressing the state in which you would like to save it in. If a user decides they would like to go back to one of their saved states, they are able to do so by pressing the load button, then pressing the state in which they would like to see displayed by the lights. A user can save at most three states at a time, and they may modify the states by pressing save and overwriting the old settings of the desired state.

1.2.2 Brightness Level

A user can change the brightness level of the lights by interacting with the QSlider implemented to change brightness levels between 1 and 255. This feature is applicable to most of our designs, excluding the ‘breathe’ option as this light pattern changes the brightness of the lights itself.

1.2.3 Accessibility Feature

User accessibility features have been implemented to let the user know what pattern option they have chosen upon selection. This feature was possible by including the QMediaPlayer class offered by QT. Upon clicking any button, a .wav file will play with the corresponding button name output through the device connected to the HDMI cord.

(ii) Python script to manipulate the lights.

1.2.4 Solid Colours

This feature changes the colour of the LED lights to correspond to a specific colour based on the red, blue, and green options presented to the user in the Qt controller. Each solid colour python script has a function called 'colorWipe' which allows all the colours in the LED strip lights to change at one time. The colour is then specified in the main, such that the desired colour is displayed based on user selection.

1.2.5 Breathe

Manipulates the lights to operate in a 'breathing' fashion, i.e. a distinct colour brightens, then dims until off and switches to a new colour and cycles again. The function 'colorWipe' is used to change the brightness level of all the LED lights in a uniform fashion.

1.2.6 Traffic Route

Emulates a four-way traffic light by looping through the different states of a traffic light.

1.2.7 Police Chase

Flashes red and blue colours and plays a police siren sound.

1.2.8 Rainbow

Cycles through all colours of the rainbow across every LED.

1.2.9 Party

Flashes random colours across all LEDs.

2.0 Key Accomplishments

2.1 What Went Right?

Overall, the project was a success. We accomplished making a working traffic light along with several other functioning light patterns. We were able to successfully achieve communication between the Raspberry Pi and the Qt Controller with the help of text files that they could both interact with. This was useful for communicating brightness to the python scripts and when a new button was pressed in order to signal to the python script to end the previous button.

2.2 What Worked Well?

The control we had over the LEDs was very helpful. We originally thought we would only be able to change the LEDs certain colours, and in blocks of 3-5, but were happy to discover we could change them individually to any colour. This allowed us to make really cool and unique patterns like rainbow, party, and police chase. The traffic light routine was the most difficult routine to make because it was the only pattern that required us to turn on/off specific LEDs, but having the complete control helped.

The controller functionality also worked well. Implementing states was challenging but worked exactly as intended. We were able to save both the brightness, and the light pattern to any of the 3 states. The brightness slider worked well to control the brightness of the lights. The text box at the top of the controller also worked well to show the user what pattern and brightness was currently being played.

2.3 What Was Found to be Particularly Useful?

The breadboard was incredibly important for the completion of this project. We found very useful tutorials for wiring and set ups involving LED strip lights that helped us create a functional circuit.

We also struggled at first to pass arguments to the python scripts from the Qt controller. This is because the QObject, QProcess did not have a feature to pass arguments to the script it was calling. We needed arguments for two reasons:

1. We needed to know the brightness.
2. We needed to know when another button was clicked in order to stop the python script.

So, we thought of a creative way to pass arguments: read and write to text files that the Qt controller and python scripts could access.

2.4 What Design Decisions Contributed to The Success of the Project?

Having two processes running and passing information to the LED strip at the same time would cause the LEDs to malfunction. If that happened, we would have to restart the Pi in order to reset the LEDs.

Passing arguments to the python scripts via text files was a huge saviour. This technique was super useful because we almost gave up on looping the light patterns. We almost said, "Let's make users run a full light pattern and force the user to wait until it is finished before they can click something else.". This seemed to be the only way we could ensure two processes don't occur at the same time, but when we implemented the text file technique, we made it possible for the user to switch patterns at any point or let the same one run endlessly instead of only once.

This also helped with testing because we no longer had to restart the Pi when something went wrong. We simply clicked the "Off" button on the controller for example, and this would terminate any existing light pattern.

3.0 Key Problem Areas

3.1 What Went Wrong?

Most of the features in our traffic light lamp were a success, however there were some features that we hoped to complete that our team was unsuccessful in implementing. Our wishlist points including a microphone were not successful, as the lights we had purchased for this project were not compatible with any audio input. In addition, our group used Qt to develop the controller that was used to manipulate the lights. Unfortunately, Qt was incompatible with audio input for the purposes we intended to use it for and had not yet implemented any voice recognition, so it was difficult for us to come up with an alternative method to implement these functions in our project.

The most challenging aspect of our project was getting the Raspberry Pi to cooperate with our Qt controller software. Because this project was a group effort, at times group members would implement portions of our controller on their own device, but when it came to testing the software on the Raspberry Pi the working software ran into issues for unforeseen reasons. Because of this, some design patterns we implemented in our controller needed to be taken out so that the core features of our project were able to function as intended.

3.2 What Project Processes Didn't Work Well?

The worst part of working on the Pi was that one person kept it at their house and no one was able to work on it or test new code without it. This massively slowed progress and forced us to have full work days over the weekends to finish huge chunks of the project at a time.

3.3 What Technical Challenges Did You Encounter?

Despite our best efforts, the first issue we ran into was implementing the microphone which we intended to manipulate the lights via audio input. Due to lack of research and resources, we found that this was not possible to implement with the LED lights that we had purchased for our traffic light. Instead, our group decided to implement a new feature using audio output for an added accessibility feature. We ran into some challenges when we were implementing the audio output of our project. The Raspberry Pi would not output sound through the TV we had hooked it up to despite us trying different HDMI cables and going through the audio config, citing that there was no volume control.

After resolving the previous issue, we decided to use the Qt's QMediaPlayer class to implement audio output corresponding with the option selected on the controller, such as police sirens playing during the police light pattern. We experienced several issues with this class as we were attempting to use it. More specifically, QMediaPlayer was not recognized, and we had trouble finding out why and how to fix it. In addition, there was very little documentation we could find on certain QMediaPlayer functions and slots, making it difficult for us to both implement and debug our audio output code.

We also experienced issues with terminating old processes when changing the state of the current process running. For example, if we changed the current state of the traffic light to display one pattern and in the case that the previous pattern was not done terminating by the time we had switched the state, the program would end up breaking or the lights would start to behave not as intended. This would result in us having to manually restart the pi everytime we wanted to test out our program.

One last issue to mention is that the colour style for our Qt controller never worked, and the controller remains grey. However, this is a very minor problem, as the controller is functional without it and is still relatively pleasing to the eye. One could argue that this improves the user-friendliness of the controller, as the interface style is simpler and there is less going on.

3.4 What Design Decisions Made it More Difficult to Succeed in Your Project?

The largest design decision that we made that made it difficult for our team to succeed in our project was our lack of inital research of c++ and the LED lights we purchased. Our team was very focused on making sure that our project was functional, that we forgot about the most important aspect of the assignment which was making sure the project was implemented in a very object-oriented way. A big part of the blame falls on our initial research, as we did not account for the amount of python that we would require to manipulate the LED lights we had purchased, which caused our UML diagram to be very simple. By the time we had realized our mistake, it was too late to change anything as assignment deadlines were around the corner. In addition, because of our lack of prior research in c++, we found it difficult to implement design patterns to our already developed program given our time constraint. If we had done more research to begin with on c++ design patterns, incorporating them could have been more possible.

3.5 What Were The Effects/Impact of These Problem Areas?

There was one positive impact. We completed the project in a way that made most sense to us because we didn't force design patterns that were not necessarily needed. This helped us move along with our project, like they say, "Done is better than perfect.".

It did, however, cost us some efficiency in areas that we could have consolidated. For example, we have several python scripts in separate files. We could have put them all into one file, made a class that embodies the LED strip, and called the corresponding functions when buttons were clicked on the controller. This idea came to us too late and would have required a massive overhaul of our existing code.

3.6 What Corrective Actions Did You Take to Resolve The problems?

We ended up using a Bluetooth speaker to resolve the issue with the Raspberry Pi not producing audio output through the TV. We first combed through the Raspberry Pi's config and the TV's settings to see if there was anything we needed to adjust. We also tried switching the HDMI cable connecting the Pi to the TV, in case the one we were currently using was not capable of audio.

Our Qt did not recognize the QMediaPlayer class, so we did some research on the error it produced to fix it. There were many online solutions, which mostly pointed out that we did not include the required line in our .pro file, and we had not installed the correct packages to use QMediaPlayer. After adding the required line, installing several packages, and figuring out that we needed to restart the Raspberry Pi to use those packages, we were able to successfully use the QMediaPlayer class.

In order to overcome our issue related to terminating processes we used a text file called runFlag that stores a 0, or 1. When a python script reads this flag as 1, it continues to run. Once it reads 0, it will end. When a button is clicked in the controller, this flag is first set to 0, terminating any pre-existing python script, and then set back to 1 before the next script is called.

Some of the issues we faced we were unable to correct. To account for the lost user points, we added new features to our project. For instance, despite our best efforts to incorporate the microphone, we were not successful in doing so. Instead, we decided it was best to replace that feature with the accessibility feature as well as the addition of some new patterns that were not originally specified in our outline or user stories. Also, we were never able to fix the issue where the colour styling of our controller would not work, but after spending a good amount of time trying to fix it, we decided to allocate our time elsewhere since this did not detract from the controller's functionality or user interface.

4.0 Lessons Learned

Overall, this project was a great learning experience for our team as this was the first time any of us had integrated both hardware and software in a project. We were able to learn a lot about the Raspberry Pi's potential, which made this project enjoyable to work on. With that being said, there are many aspects of this project that we would do differently. One of the greatest struggles we encountered while developing our project was finding ways to make it more object-oriented. When we first came up with our idea, we did not anticipate the amount of python we would need to develop the light patterns for the LED lights that we had purchased. Because of this, our UML diagram appeared very simple, though our project was quite complex in its implementation. In addition, the number of hours accounted for each user story was not accurate. We were very optimistic during our planning stages and did not properly anticipate the amount of work that goes into using a Raspberry Pi. For future projects, it may be beneficial to dedicate more time towards the specifics during the planning stages in order to further improve the design of future projects.

A good take away of the project was the way we went about implementing the features. In order to avoid communication issues, the majority of our final stage was worked on together in person. Because of our group members' good teamwork and cooperation, we were able to bounce ideas off each other, making it much easier to develop software that worked well to manipulate our hardware. While developing our controller, we struggled to initially come together to implement it due to lack of communication, which made it difficult for everyone's contribution to make an appearance in the code because of overlapping work. We quickly recognized this issue, as we decided to work on the remainder of the project together which ended up being a great decision and something that will definitely be kept in mind for future projects.

During this project, we were able to learn so many different processes that will be applicable to future endeavors. Almost all of us were new to using the Jira ticketing system, which we quickly learned is a great tool to back track your work and present new ideas in a group setting. In addition, other processes that would work well for our group were the weekly meeting times we set up with group members to track the progress of listed tasks. Keeping each other accountable and working together on tasks during the semester is what made our project successful.

A project like this one is something we would all be interested in taking part in again. Creating different patterns and learning how to manipulate the lights using a Raspberry Pi was very rewarding, perhaps in the future more complex patterns could be implemented to make for an even better and more rewarding project.