SMART HOLIDAY LIGHTS APPLICATION

Project Report

TEAM MEMBERS
Pearl Olanrewaju, Chloe Fleming, Anusha Srivatsa

Course Title

ECE 558: Embedded System Programming

Overview

The Smart Holiday Lights app brings Christmas lights into the Internet of Things! This app will allow the user to program a string of LEDs with a variety of colors and patterns from their Android phone, making holiday decorating very customizable. The Android device is able to communicate via Bluetooth with the Arduino microcontroller powering the lights. Once a Bluetooth connection is established, the user is able to select different light patterns and also able to customize the color of the lights. The user has the capability of choosing between six different patterns, four of which can be displayed in any color selected by the user. The range of color customization is endless because each of the Red, Green and Blue (RGB) components has 256 different color intensities.

Figure 1 (below) shows the very first activity that appears upon launching the app, which gives the user the ability to connect to the Arduino board via Bluetooth. Once the connection is established, it brings up the next activity that allows the user to customize the digital LEDs, as shown in Figure 2.

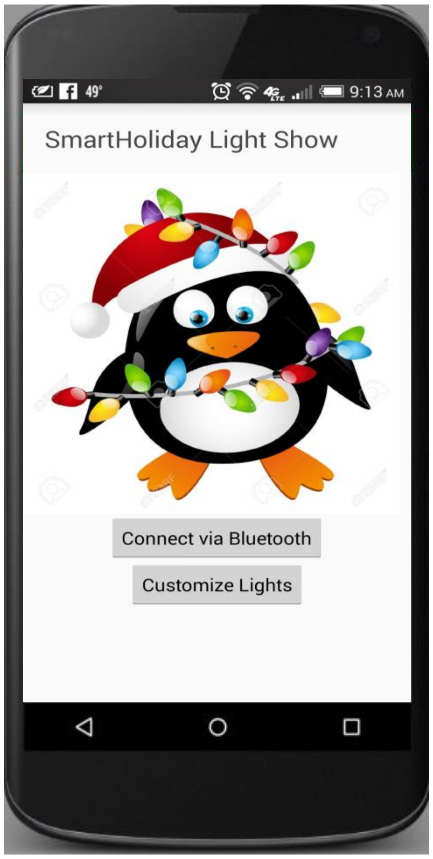
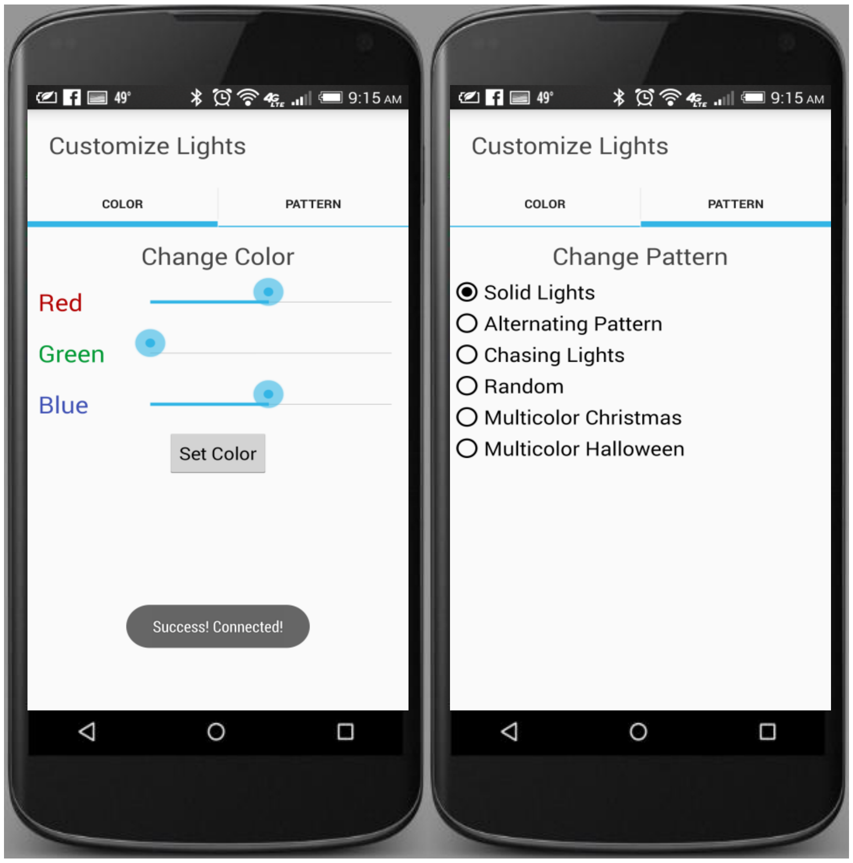


Figure 1: MainActivity



(● Solid Lights) ○ Alternating Pattern ○ Chasing Lights ○ Random ○ Multicolor Christmas ○ Multicolor Halloween

Figure 2. CustomizationActivity with tabs for changing LED color and pattern

Design Details

Hardware Components

To be able to implement the Smart Holiday Lights Application, our team used the following components:

- Android Phone
- Arduino Uno Board
- Kedsum Arduino Bluetooth Transceiver
- Adafruit Neopixel Light Strip
- 9V Battery
- Miscellaneous Wires, Resistors, and a Breadboard

The software required to implement this project falls into two different categories:

- The Android application, consisting of Java and XML produced with Android Studio.
- The light controller program for the Arduino, written as C-style code in the Arduino IDE.

We elaborate further on both software components in the sections below.

Android Application View/Controller Components

The Smart Holiday Lights app consists of the two separate activities described below.

1. MainActivity

This activity is visible as soon as the app is launched, and the main purpose of this view is to enable the user to establish a Bluetooth connection between the Android device and the Arduino. The UI of this activity consists of a cute, festive drawable and two buttons:

- **The Bluetooth button** performs most of the "heavy lifting" involved in this app. It first checks whether Bluetooth is enabled, and if not starts an Intent that will ask the user for permission to turn it on. As soon as this intent returns, an AsyncTask will be created to open a Bluetooth connection. This AsyncTask shows a ProgressBar, searches for paired devices, opens a socket to the Arduino's Bluetooth, calls a static method of CustomizationActivity that will set a handle to the Bluetooth stream, and then automatically launches the CustomizationActivity. If any one of these steps fails, the app will show an appropriate toast message to indicate that the user should try again to connect.

- **The Customize button** can also used to launch the CustomizationActivity once a Bluetooth connection is established. This button is used in the case that the user ends up back in the MainActivity again after connecting (if they accidentally press the back button, etc.), so that they will be able to get to the customization menu again without having to re-connect to Bluetooth.

2. CustomizationActivity

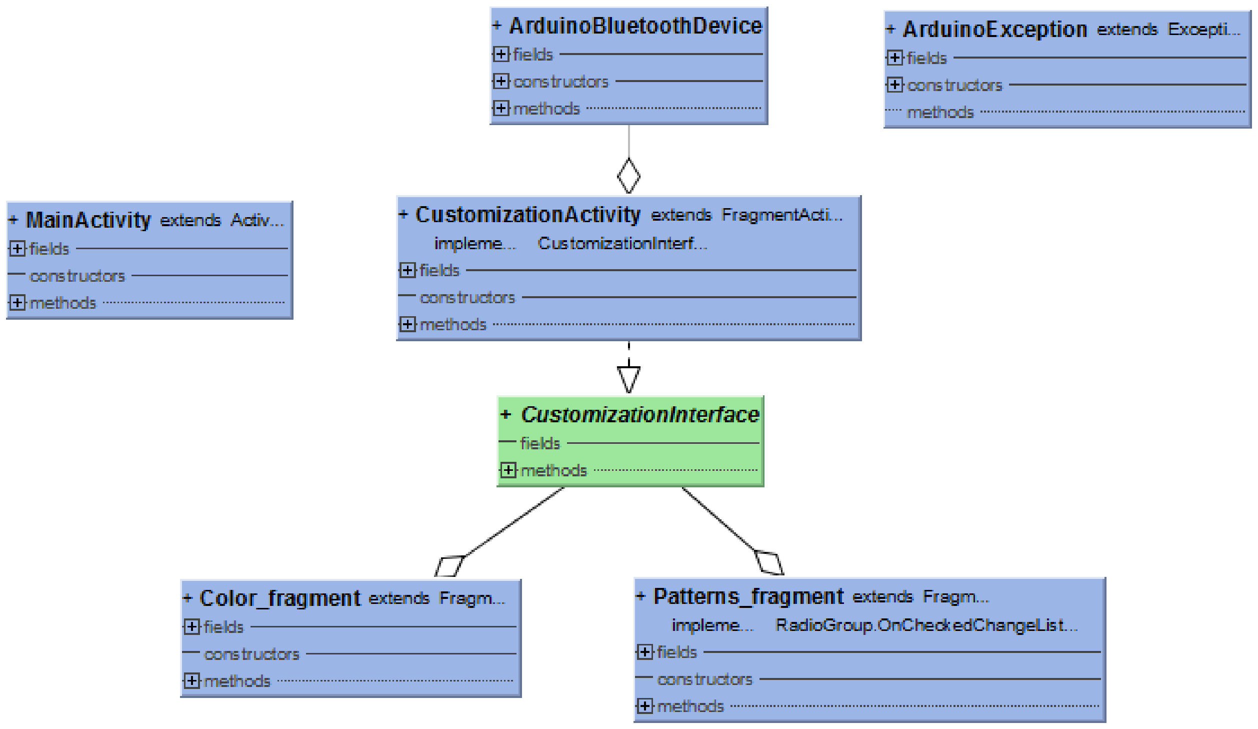
This activity host 2 fragments: one for color customization and one for pattern customization. We decided to embed both fragments in a FragmentTabHost widget, which adds seamless navigation between the two menus and adds some sophistication to the design. This activity also handles communication with the Arduino using the Bluetooth connection passed to it by MainActivity. In order for the fragments' UI handlers to easy call back to the activity and trigger Bluetooth commands, we designed a CustomizationInterface that the CustomizationActivity implements with methods setColor() and setPattern().

- **The Color fragment** allows customization of the colors of the LED using a SeekBar to set each RGB color component. The value range for the RGB colors is set from 0 to 255, to match the Neopixel specs and give the users a full range of colors. We implemented the SeekBar's onProgressChanged() method to update state variables to store the desired red, green, and blue color levels. After the user selects the color values he/she wants for the lights, pressing a button will apply the color by calling CustomizationActivity's setColor().
- **The Pattern fragment** allows customization of the LEDs using different sets of pre-defined patterns. We used a ScrollView to display a list of RadioGroup buttons, which allows the user to select only one pattern at a time. We used the onCheckedChanged() that will be triggered when the selected RadioButton changes to call CustomizationActivity's setPattern().

Android Application Data/Model Component

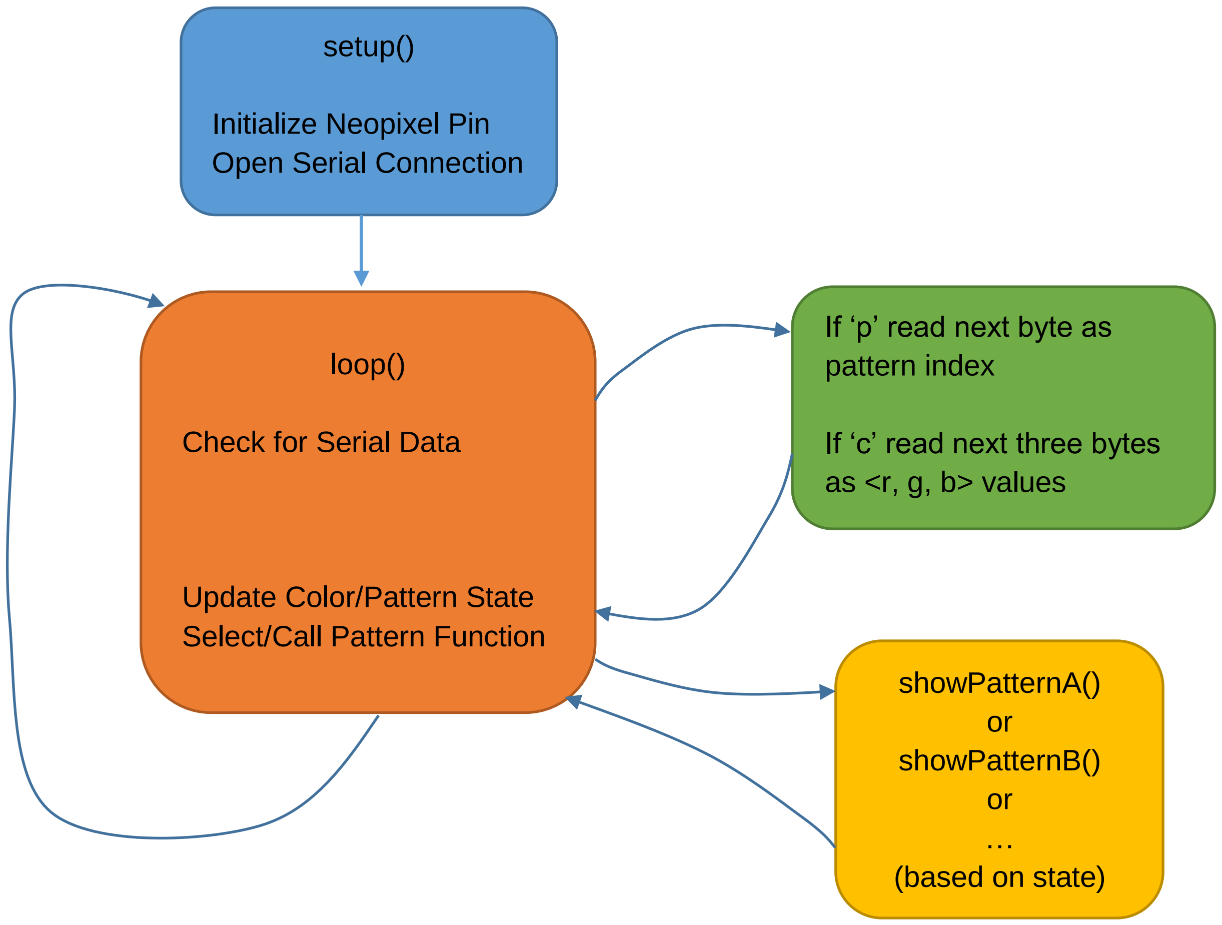
We decided that the best way to design an interface for writing the Bluetooth commands to the Arduino would be to encapsulate the functionality in a class called ArduinoBluetoothDevice. This class is instantiated with a BluetoothSocket object that is opened by the Bluetooth management API, which it uses to open an OutputStream object. The class implements methods for writing correctly formatted bytes to this stream to match the command sequences expected by the Arduino. To accompany this class, we also defined a custom ArduinoException class that derives from Exception that we could use to indicate when data mismatch or other error occurred. The CustomizationActivity maintains a static ArduinoBluetoothDevice reference and a public method call for setting this up, which is called by the MainActivity once the connection is secured.

UML Diagram of Android Application


+ ArduinoBluetoothDevice
fields
constructors
methods
+ ArduinoException extends Excepti...
fields
constructors
methods
+ MainActivity extends Activ...
fields
constructors
methods
+ CustomizationActivity extends FragmentActi... impleme... CustomizationInterf...
fields
constructors
methods
+ CustomizationInterface
fields
methods
+ Color_fragment extends Fragm...
fields
constructors
methods
+ Patterns_fragment extends Fragm... impleme... RadioGroup.OnCheckedChangeList...
fields
constructors
methods

Arduino Code

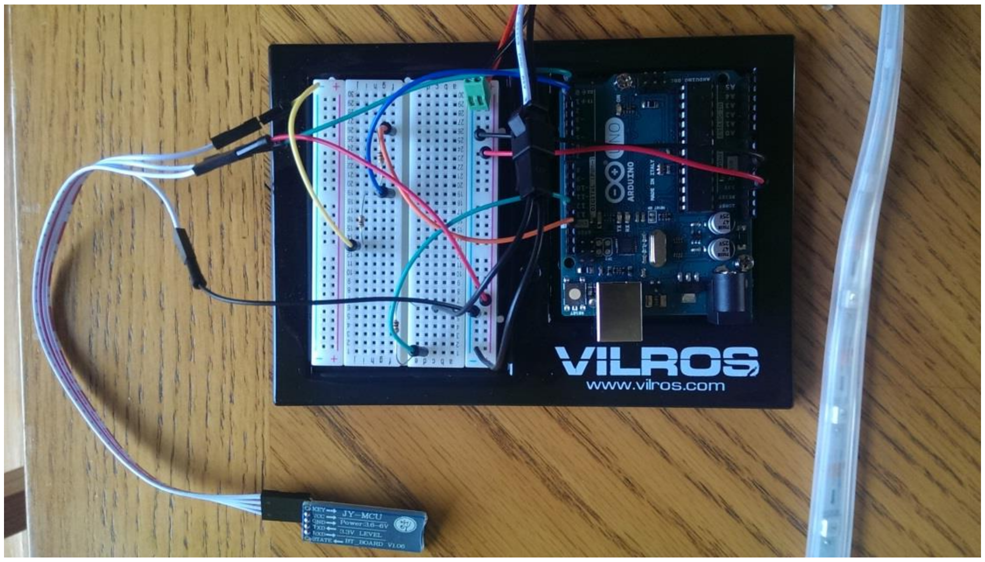
In order for the Arduino to effectively manage the lights as needed by this project, it must be responsible for three things: receiving commands sent over Bluetooth (which is routed through the board's serial Tx/Rx pins), updating color and pattern state variables, and programming the Neopixel appropriately. We used Arduino programming conventions and libraries, which allow for simple C-style code implemented in setup() and loop() functions. We implemented six additional functions, each of which lights up the Neopixel in a different pattern using loops, delays, color state variables, and a random number generator. The program logic is summarized by the flow chart below.


setup()
Initialize Neopixel Pin
Open Serial Connection
loop()
Check for Serial Data
Update Color/Pattern State
Select/Call Pattern Function
If 'p' read next byte as pattern index
If 'c' read next three bytes as <r, g, b> values
showPatternA()
or
showPatternB()
or
…
(based on state)

Setup and Usage

Given all of the software and hardware components described above, use the following procedure to deploy your own Smart Holiday Lights project:

1. Wire the Bluetooth transceiver and Neopixel to the breadboard, which in turn should be connected to power and GPIO pins on the Arduino. Below is an image of our wiring for reference. We connected the Neopixel to GPIO pin 12, but this is not a hard requirement as long as the #define in the Arduino code is updated to reflect the pin that it is actually connected to.



2. After installing the Arduino IDE, set up the build environment for the Arduino Uno and load our code file (.ino) onto the device using a USB cable.
3. Apply power to the Arduino Uno and ensure that all connections are secure. At this point the Neopixel should draw the “Chasing Lights” pattern in green, and the Bluetooth transceiver will have a flashing red light to indicate that it is not yet paired.
4. Deploy the Smart Holiday Lights app onto an Android device using Android Studio or adb. Launch the app, press the “Connect via Bluetooth” button, and confirm that the app is allowed to use Bluetooth when the Intent dialog appears. The first time a new device is used with the Arduino Uno, the user will first need to go to the Bluetooth menu in their device settings and manually pair with it. The transceiver will show up as HC-06 and pairs with pin 1234.

5. Once connected, the Bluetooth transceiver light will change to a solid red. Proceed to customize your lights and enjoy the results!

Results

We successfully achieved our goal of creating an app that could program the LEDs to different patterns and colors! We filmed a short demo and posted it on Youtube: https://www.youtube.com/watch?v=9bTaO92Tq1Y

Contributions of Team Members

- Chloe wrote the Arduino code, as well as testing the functionality of the code written by the other team members, implementing the AsyncTask, and cleaning up the UI.
- Pearl did most of the app UI and structure setup, creating both activities and the fragment managers, and XML. She also implemented the code behind the pattern fragment.
- Anusha worked on the Bluetooth connectivity and also implemented the code behind the color fragment.

Issues Faced

One of our biggest issues faced during this project was being able to test our progress. We had one hardware setup, so we frequently had to send code back and forth to be tested. We were able to make it work by constantly keeping in touch with each other on progress and also by working on one common code using git repositories. We also met frequently to test core functionality such as Bluetooth connectivity together, before dividing up feature work that depended on this functionality.

We took a while to get the Bluetooth going, figuring out how to properly implement it and handling exceptions. We also wanted to be able to give some feedback to the user that the connection was established, which was tricky given that we had been spawning a background thread to complete the connection. We finally were able to accomplish this by researching and implementing an AsyncTask that interacted with a ProgressBar widget.

We ran into a few issues in using the SeekBar to control color settings. Originally we wanted to be able to change color real time from the slider, so that the user could see the color change just by dragging the slider thumb. But it was not as seamless as we expected considering the amount of data traffic that was to be sent to the Arduino: onProgressChanged() ended up being called for every unit shifted and thus triggering a Bluetooth command. Each color draw loop can take up to few seconds to draw, depending on the delay inherent in the pattern, so the user experience was poor with that implementation. Instead we added a button that sent color values only when the user clicked on it.

The most frustrating and initially mysterious issue we faced was that at times the Arduino would randomly power cycle during our testing. We at first thought that this might be because our program was using too much RAM (since the Neopixel requires a large amount of memory to store data about each LED), but eventually narrowed it down to a power flow issue with the wall adapter or a drained battery. When we used a more reliable power source such as a fresh battery, the issue disappeared.

Ideas for Future Work

1. More creativity in the number of patterns, perhaps including the ability for the user to be able to create brand new patterns or select how individual LEDs should be set.
2. Decoupling interface between Android and Arduino, so that the pattern list is not hardcoded to match the Arduino program. Perhaps the Arduino could enumerate its pattern capabilities to the Android device upon connection, etc. and the app could receive this data and populate the pattern list dynamically.
3. Integration of more Android and Arduino capabilities to add features such as reminders. For instance, the Arduino could host a light sensor and post a notification to the Android device when it becomes dark enough to turn on your lights, etc.