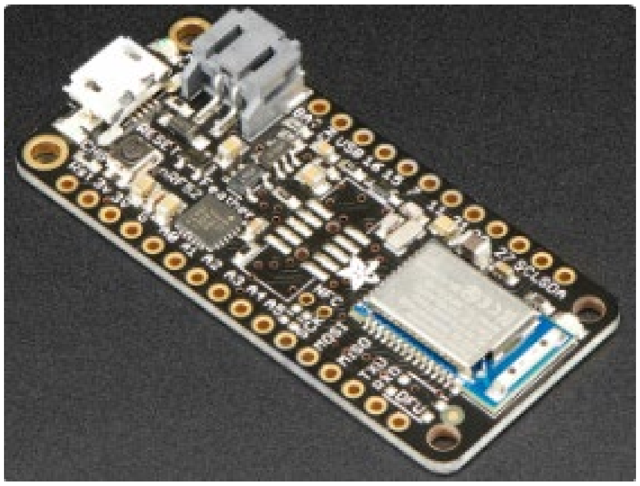
Junior Design 1195: Design Project 3

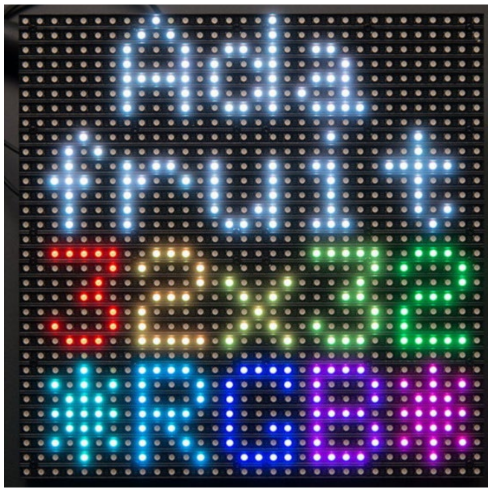
Zach DeGore

Design Overview

This project was intended to be an app that had functionalities that controlled an LED display over Bluetooth with an NRF52 feather chip. The app has separate pages each having a different "mode" to control, with different adjustments to control on each page. Based on the selections made on the app it would be transmitted over Bluetooth to the Arduino chip which would handle controlling the LED display. The app is programmed in swift which is a language that I have never used before. It is the language and software used by Apple for their apps, I wanted to learn how to create an app using this language, as well as have it control something. This project was probably something that should have been completed over a longer period than 6 weeks as certain parts I misjudged the difficulty and would have been easier with more time. After testing I needed to switch the design as the original would not work. I decided to instead control an LED strip, which could still use the different pages for different modes, in a slightly different manner than originally intended. I also ordered a chip that was able to drive the LED display, but the main goal was LED strips. This was a time crunch as the strips arrived during finals week and required me to gut a lot of the code I already had.

Left shows NRF52 feather chip, right 64x64 Adafruit display





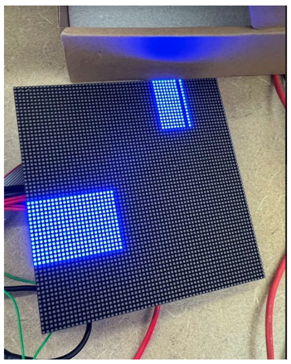
Preliminary Design Verification

Initial Testing

I spent a lot of time in this phase trying to get components to function together which in the end did not work. Firstly, the Arduino chip which is an NRF52 feather. It took about a week and a half to understand the setup of the chip and pins as well as using the base app that came with it to provide basic controls, and to setup Bluetooth scanning, connection, and receiving. I tested controls using the pins and making sure each pin was able to output as well as using the serial monitor to check reading In from Bluetooth. I also needed to test outputting to the LED display which used 16 pins.

Problems

The main issue arrived when trying to get the LED display to work with the feather chip. There are many Arduino libraries for controlling LED displays, each varying in capabilities. The display I chose was 64x64 bit, this limited the available libraries that could be used as only some were written to handle 64x64 as it used 5 addressing bits instead of the typical 3 or 4 bits. It was then even further limited because of the chip I was using NRF52 feather. After trying many different libraries, it took many tries to even get the display to output something, this something happened to be nothing, it would only output random patterns and lights as seen below.



Solutions

After about 2 weeks of trying, I figured the board didn't have the capabilities to run the matrix. After putting so much time in trying to get the display to work it put me behind a great deal. I came up with two alternatives to at least have something work. One, switch the display for LED strips, and two switch the chip for a stronger one that would be able to drive the display.

Design Implementation

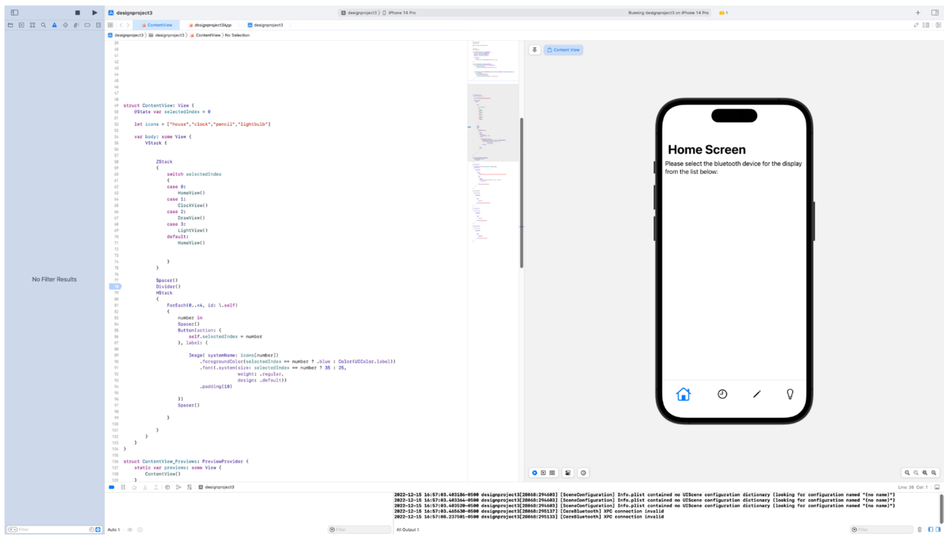
The Code is split between the app code and Arduino code, Arduino being responsible for controlling the LED lights based on the app inputs from Bluetooth. The app has some basic controls on each page that would allow for customizations of the different modes. The app has four pages, Bluetooth, solid color mode, train mode, and fade mode. The first page Bluetooth is responsible for all things related to Bluetooth, second solid color mode allows the user to select a color to be shown on the LED strip, third train mode allows the user to pick a pattern, fade type brightness and speed, and lastly fade mode which allows the user to pick the brightness and speed of the fading colors. The app code unfortunately is only able to be opened on MAC because it uses XCode, so a lot of the code will be shown and explained below.

App Code Version 1.0

There were two versions of the app that I had used, the first had a simple tab bar at the bottom that allowed for easy switching between the different pages. After further research into Bluetooth, I realized that it needed to be implemented using ViewController, this was completely separate from what I was using so I had to start over.

View and Tab bar

This image shows the base code on the left-side for having multiple view and the tab bar at the bottom to control each page. Selecting the buttons at the bottom corresponds to a different page which was originally meant for the different clock faces/features. The right-side shows what the code would look like running on an iPhone, which updates based on the features added in the code.

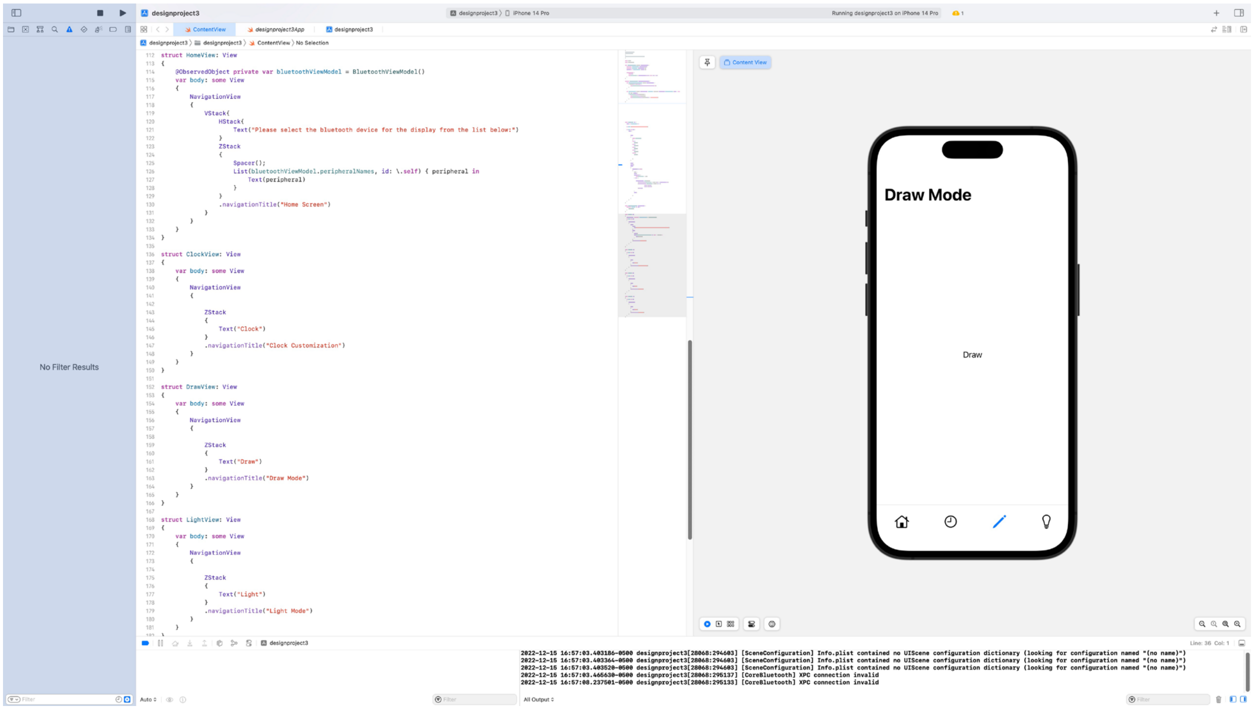


```
struct ContentView: View {
    @State var selectedIndex = 0

    let icons = ["house","clock","pencil","lightbulb"]

    var body: some View {
        VStack {


            ZStack
            {
                switch selectedIndex
                {
                case 0:
                    HomeView()
                case 1:
                    ClockView()
                case 2:
                    DrawView()
                case 3:
                    LightView()
                default:
                    HomeView()


                }
            }

            Spacer()
            Divider()
            HStack
            {
                ForEach(0..<4, id: \.self)
                {
                    number in
                    Spacer()
                    Button(action: {
                        self.selectedIndex = number
                    }, label: {

                        Image( systemName: icons[number])
                            .foregroundColor(selectedIndex == number ? .blue : Color(UIColor.label))
                            .font(.system(size: selectedIndex == number ? 35 : 25,
                                          weight: .regular,
                                          design: .default))
                            .padding(10)

                    })
                    Spacer()

                }

            }
        }
    }
}

struct ContentView_Previews: PreviewProvider {
    static var previews: some View {
        ContentView()
    }
```

```
2022-12-15 16:57:03.403186-0500 designproject3[28068:294603] [SceneConfiguration] Info.plist contained no UIScene configuration dictionary (looking for configuration named "(no name)")
2022-12-15 16:57:03.403364-0500 designproject3[28068:294603] [SceneConfiguration] Info.plist contained no UIScene configuration dictionary (looking for configuration named "(no name)")
2022-12-15 16:57:03.403520-0500 designproject3[28068:294603] [SceneConfiguration] Info.plist contained no UIScene configuration dictionary (looking for configuration named "(no name)")
2022-12-15 16:57:03.465630-0500 designproject3[28068:295137] [CoreBluetooth] XPC connection invalid
2022-12-15 16:57:08.237501-0500 designproject3[28068:295133] [CoreBluetooth] XPC connection invalid
```

Individual Views

This shows what the code for each individual view code looks like, as more code would have been added for each view it would have gone here. They are written as functions so that in the view select you can easily switch view by calling a given view function. Each view contains a navigation view which is the base for each page, then things like zstack or h/vstack hold the text fields or buttons.

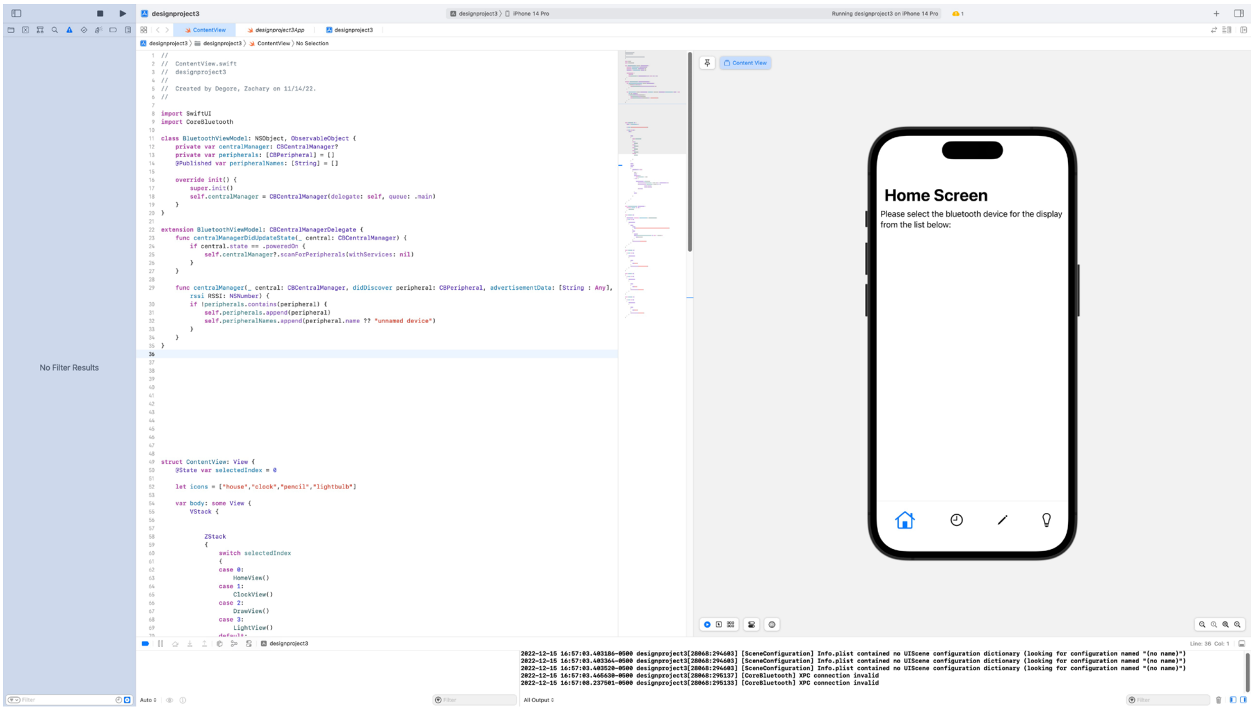


```swift
struct HomeView: View
{
    @ObservedObject private var bluetoothViewModel = BluetoothViewModel()
    var body: some View
    {
        NavigationView
        {
            VStack{
                HStack{
                    Text("Please select the bluetooth device for the display from the list below:")
                }
                ZStack
                {
                    Spacer();
                    List(bluetoothViewModel.peripheralNames, id: \.self) { peripheral in
                        Text(peripheral)
                    }
                }
                .navigationTitle("Home Screen")
            }
        }
    }
}

struct ClockView: View
{
    var body: some View
    {
        NavigationView
        {

            ZStack
            {
                Text("Clock")
            }
            .navigationTitle("Clock Customization")
        }
    }
}

struct DrawView: View
{
    var body: some View
    {
        NavigationView
        {

            ZStack
            {
                Text("Draw")
            }
            .navigationTitle("Draw Mode")
        }
    }
}

struct LightView: View
{
    var body: some View
    {
        NavigationView
        {

            ZStack
            {
                Text("Light")
            }
            .navigationTitle("Light Mode")
        }
    }
```

```
2022-12-15 16:57:03.403186-0500 designproject3[28068:294603] [SceneConfiguration] Info.plist contained no UIScene configuration dictionary (looking for configuration named "(no name)")
2022-12-15 16:57:03.403364-0500 designproject3[28068:294603] [SceneConfiguration] Info.plist contained no UIScene configuration dictionary (looking for configuration named "(no name)")
2022-12-15 16:57:03.403520-0500 designproject3[28068:294603] [SceneConfiguration] Info.plist contained no UIScene configuration dictionary (looking for configuration named "(no name)")
2022-12-15 16:57:03.465630-0500 designproject3[28068:295137] [CoreBluetooth] XPC connection invalid
2022-12-15 16:57:08.237501-0500 designproject3[28068:295133] [CoreBluetooth] XPC connection invalid
```

Bluetooth

This code is what was used to show the available Bluetooth devices. However, this does not support the option to connect or send information to a Bluetooth device, it just shows the available devices which was not enough for what I wanted to accomplish. The screen on the right would have shown a list containing all the devices that were within range, but the simulation does not support this as it has no real Bluetooth connection, but a real phone would show these values.



```swift
//
//  ContentView.swift
//  designproject3
//
//  Created by Degore, Zachary on 11/14/22.
//

import SwiftUI
import CoreBluetooth

class BluetoothViewModel: NSObject, ObservableObject {
    private var centralManager: CBCentralManager?
    private var peripherals: [CBPeripheral] = []
    @Published var peripheralNames: [String] = []

    override init() {
        super.init()
        self.centralManager = CBCentralManager(delegate: self, queue: .main)
    }
}

extension BluetoothViewModel: CBCentralManagerDelegate {
    func centralManagerDidUpdateState(_ central: CBCentralManager) {
        if central.state == .poweredOn {
            self.centralManager?.scanForPeripherals(withServices: nil)
        }
    }

    func centralManager(_ central: CBCentralManager, didDiscover peripheral: CBPeripheral, advertisementData: [String : Any],
        rssi RSSI: NSNumber) {
        if !peripherals.contains(peripheral) {
            self.peripherals.append(peripheral)
            self.peripheralNames.append(peripheral.name ?? "unnamed device")
        }
    }
}














struct ContentView: View {
    @State var selectedIndex = 0

    let icons = ["house","clock","pencil","lightbulb"]

    var body: some View {
        VStack {


            ZStack
            {
                switch selectedIndex
                {
                case 0:
                    HomeView()
                case 1:
                    ClockView()
                case 2:
                    DrawView()
                case 3:
                    LightView()
```

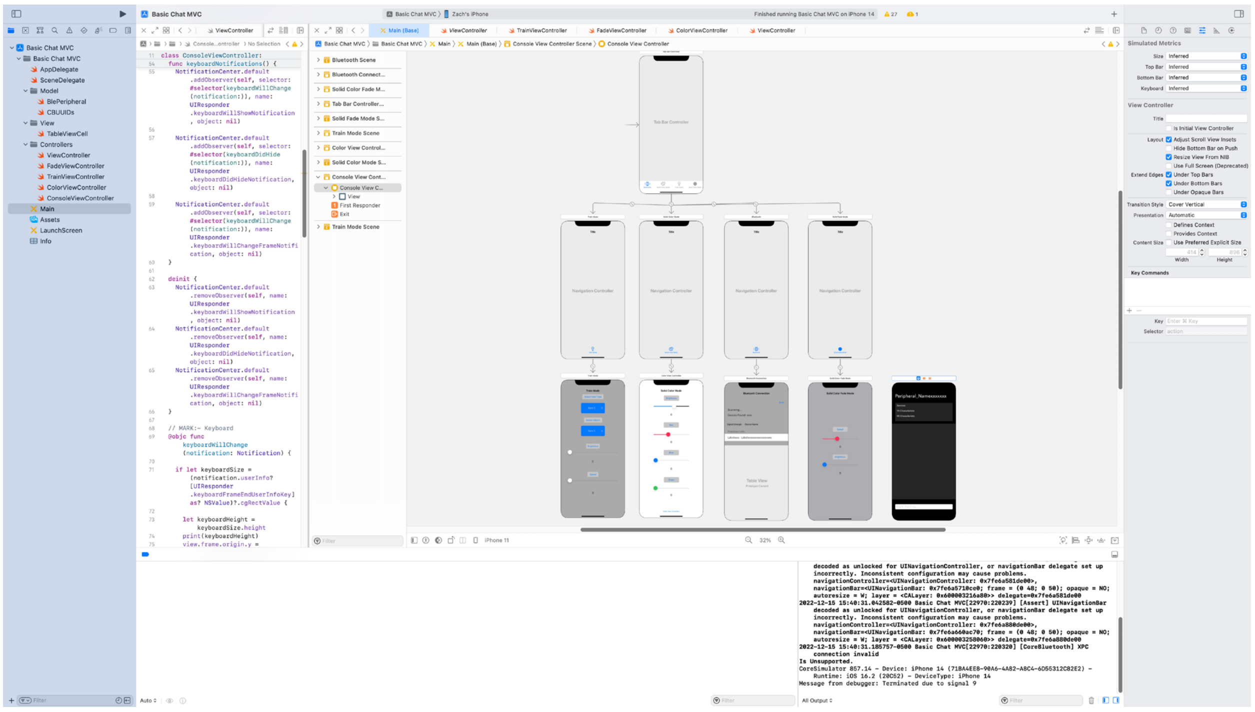
Home Screen
Please select the bluetooth device for the display from the list below:

```
2022-12-15 16:57:03.403186-0500 designproject3[28068:294603] [SceneConfiguration] Info.plist contained no UIScene configuration dictionary (looking for configuration named "(no name)")
2022-12-15 16:57:03.403364-0500 designproject3[28068:294603] [SceneConfiguration] Info.plist contained no UIScene configuration dictionary (looking for configuration named "(no name)")
2022-12-15 16:57:03.403520-0500 designproject3[28068:294603] [SceneConfiguration] Info.plist contained no UIScene configuration dictionary (looking for configuration named "(no name)")
2022-12-15 16:57:03.465630-0500 designproject3[28068:295137] [CoreBluetooth] XPC connection invalid
2022-12-15 16:57:08.237501-0500 designproject3[28068:295133] [CoreBluetooth] XPC connection invalid
```

App Code Version 2.0

Story Board

This code is shows the second iteration of the app using ViewController, this contains a sort of hierarchy, at the base/top is the tab bar controller. This controller controls which view is selected and which view to switch to based on which tab bar button is selected. This implementation also allows for the use of a story board, which is what is seen below. The story board allows for quicker customizations of things like buttons, text fields, etc., based on the controls on the right of the screen instead of coding individual fields, sizes, font, etc. The story board still requires the use of code for each view to implement functions as it is mainly visual customizations while code is functional customizations. Back to the hierarchy the tab bar controller is linked to each view by a navigation view controller which like before is the base for each view. Each navigation view is linked to the actual view which holds the text/button fields, each varying in design based on the purpose of the page.

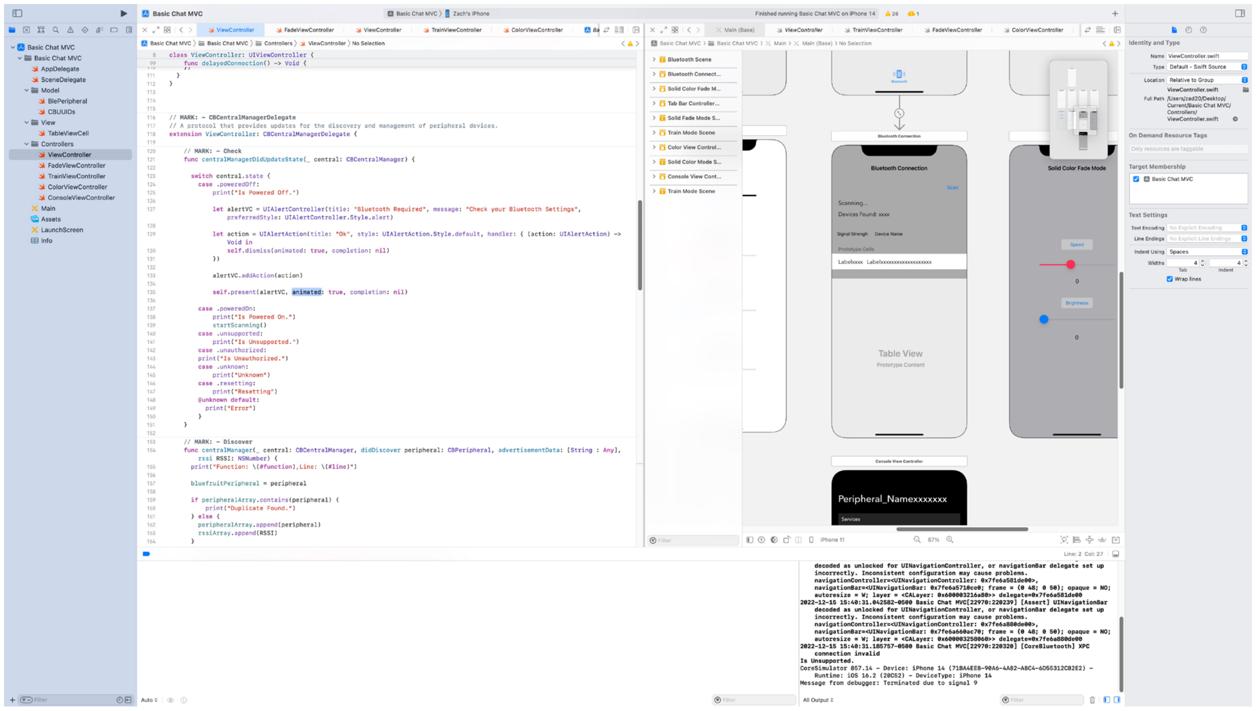


```
class ConsoleViewController:
  func keyboardNotifications() {
    NotificationCenter.default
        .addObserver(self, selector:
        #selector(keyboardWillChange
        (notification:)), name:
        UIResponder
        .keyboardWillShowNotification
        , object: nil)

    NotificationCenter.default
        .addObserver(self, selector:
        #selector(keyboardDidHide
        (notification:)), name:
        UIResponder
        .keyboardDidHideNotification,
        object: nil)

    NotificationCenter.default
        .addObserver(self, selector:
        #selector(keyboardWillChange
        (notification:)), name:
        UIResponder
        .keyboardWillChangeFrameNotifi
        cation, object: nil)
  }

  deinit {
    NotificationCenter.default
        .removeObserver(self, name:
        UIResponder
        .keyboardWillShowNotification
        , object: nil)
    NotificationCenter.default
        .removeObserver(self, name:
        UIResponder
        .keyboardDidHideNotification,
        object: nil)
    NotificationCenter.default
        .removeObserver(self, name:
        UIResponder
        .keyboardWillChangeFrameNotifi
        cation, object: nil)
  }

  // MARK:- Keyboard
  @objc func
      keyboardWillChange
      (notification: Notification) {

    if let keyboardSize =
        (notification.userInfo?
        [UIResponder
        .keyboardFrameEndUserInfoKey]
        as? NSValue)?.cgRectValue {

      let keyboardHeight =
          keyboardSize.height
      print(keyboardHeight)
      view.frame.origin.y =
```

```
    decoded as unlocked for UINavigationController, or navigationBar delegate set up
    incorrectly. Inconsistent configuration may cause problems.
    navigationController=<UINavigationController: 0x7fe6a581de00>,
    navigationBar=<UINavigationBar: 0x7fe6a5710ce0; frame = (0 48; 0 50); opaque = NO;
    autoresize = W; layer = <CALayer: 0x600003216a80>> delegate=0x7fe6a581de00
2022-12-15 15:40:31.042582-0500 Basic Chat MVC[22970:220239] [Assert] UINavigationBar
    decoded as unlocked for UINavigationController, or navigationBar delegate set up
    incorrectly. Inconsistent configuration may cause problems.
    navigationController=<UINavigationController: 0x7fe6a880de00>,
    navigationBar=<UINavigationBar: 0x7fe6a660ac70; frame = (0 48; 0 50); opaque = NO;
    autoresize = W; layer = <CALayer: 0x600003258060>> delegate=0x7fe6a880de00
2022-12-15 15:40:31.185757-0500 Basic Chat MVC[22970:220320] [CoreBluetooth] XPC
    connection invalid
Is Unsupported.
CoreSimulator 857.14 - Device: iPhone 14 (71BA4EE8-90A6-4A82-A8C4-6D55312C82E2) -
    Runtime: iOS 16.2 (20C52) - DeviceType: iPhone 14
Message from debugger: Terminated due to signal 9
```

In depth look at a view controller(Bluetooth page)

This shows the corresponding code that goes with the Bluetooth view. This page has over two hundred lines of code for Bluetooth broadcasting, discovery, connection and sending/receiving. Unfortunately, this code does not work correctly, it is from an Adafruit tutorial specifically for the Bluetooth chip I used(NRF52 feather). I believe that the issue is something with the UUID which specifies which type of device to search for. In this case nothing would show as a result of searching for Bluetooth devices which leads me to believe that an incorrect UUID was used but I did not have time in the end to attempt to correct this. Continuing to look at the right-side of the screen it shows something called a table view which allows you to have different lines of information with each varying based on what would have been the device strength and name for discovered Bluetooth devices where two labels under Signal Strength and Device Name.

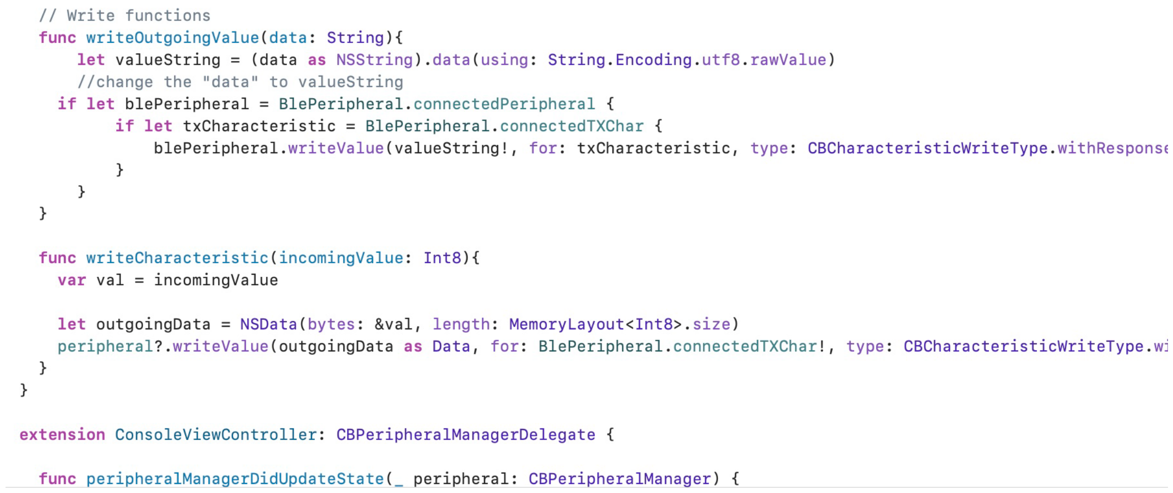


```
class ViewController: UIViewController {
    func delayedConnection() -> Void {
    }
}



// MARK: - CBCentralManagerDelegate
// A protocol that provides updates for the discovery and management of peripheral devices.
extension ViewController: CBCentralManagerDelegate {

    // MARK: - Check
    func centralManagerDidUpdateState(_ central: CBCentralManager) {

      switch central.state {
        case .poweredOff:
            print("Is Powered Off.")

            let alertVC = UIAlertController(title: "Bluetooth Required", message: "Check your Bluetooth Settings",
                preferredStyle: UIAlertController.Style.alert)

            let action = UIAlertAction(title: "Ok", style: UIAlertAction.Style.default, handler: { (action: UIAlertAction) ->
                Void in
                self.dismiss(animated: true, completion: nil)
            })

            alertVC.addAction(action)

            self.present(alertVC, animated: true, completion: nil)

        case .poweredOn:
            print("Is Powered On.")
            startScanning()
        case .unsupported:
            print("Is Unsupported.")
        case .unauthorized:
        print("Is Unauthorized.")
        case .unknown:
            print("Unknown")
        case .resetting:
            print("Resetting")
        @unknown default:
          print("Error")
        }
    }

    // MARK: - Discover
    func centralManager(_ central: CBCentralManager, didDiscover peripheral: CBPeripheral, advertisementData: [String : Any],
        rssi RSSI: NSNumber) {
      print("Function: \(#function),Line: \(#line)")

      bluefruitPeripheral = peripheral

      if peripheralArray.contains(peripheral) {
          print("Duplicate Found.")
      } else {
        peripheralArray.append(peripheral)
        rssiArray.append(RSSI)
      }
```

```
    decoded as unlocked for UINavigationController, or navigationBar delegate set up
    incorrectly. Inconsistent configuration may cause problems.
    navigationController=<UINavigationController: 0x7fe6a581de00>,
    navigationBar=<UINavigationBar: 0x7fe6a5710ce0; frame = (0 48; 0 50); opaque = NO;
    autoresize = W; layer = <CALayer: 0x600003216a80>> delegate=0x7fe6a581de00
2022-12-15 15:40:31.042582-0500 Basic Chat MVC[22970:220239] [Assert] UINavigationBar
    decoded as unlocked for UINavigationController, or navigationBar delegate set up
    incorrectly. Inconsistent configuration may cause problems.
    navigationController=<UINavigationController: 0x7fe6a880de00>,
    navigationBar=<UINavigationBar: 0x7fe6a660ac70; frame = (0 48; 0 50); opaque = NO;
    autoresize = W; layer = <CALayer: 0x600003258060>> delegate=0x7fe6a880de00
2022-12-15 15:40:31.185757-0500 Basic Chat MVC[22970:220320] [CoreBluetooth] XPC
    connection invalid
Is Unsupported.
CoreSimulator 857.14 - Device: iPhone 14 (71BA4EE8-90A6-4A82-A8C4-6D55312C82E2) -
    Runtime: iOS 16.2 (20C52) - DeviceType: iPhone 14
Message from debugger: Terminated due to signal 9
```

Bluetooth Sending

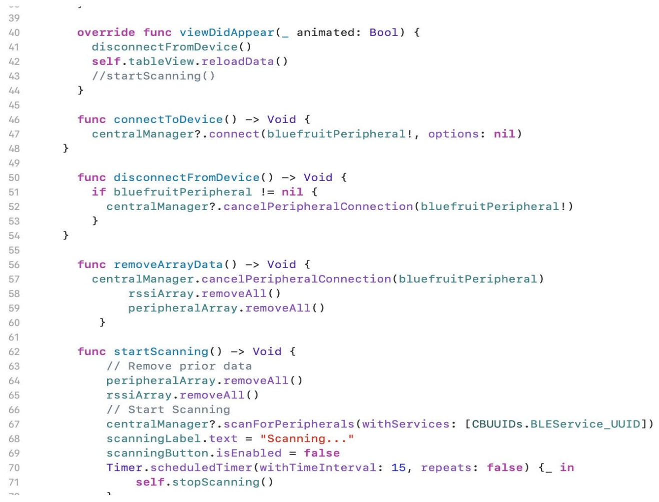
If the Bluetooth would have function correctly this function shows what sending information would have looked like. For the Arduino code which will be discussed later the values for things like color, brightness, etc., are sent as numbers separated by spaces for each value. So, sending the values selected in the views would have been appended together separated by spaces and sent to writeCharacteristics(int) so that the NRF52 feather would receive the values as an int and be able to read the values.



```
  // Write functions
  func writeOutgoingValue(data: String){
      let valueString = (data as NSString).data(using: String.Encoding.utf8.rawValue)
      //change the "data" to valueString
    if let blePeripheral = BlePeripheral.connectedPeripheral {
          if let txCharacteristic = BlePeripheral.connectedTXChar {
              blePeripheral.writeValue(valueString!, for: txCharacteristic, type: CBCharacteristicWriteType.withResponse
          }
      }
  }

  func writeCharacteristic(incomingValue: Int8){
    var val = incomingValue

    let outgoingData = NSData(bytes: &val, length: MemoryLayout<Int8>.size)
    peripheral?.writeValue(outgoingData as Data, for: BlePeripheral.connectedTXChar!, type: CBCharacteristicWriteType.wi
  }
}

extension ConsoleViewController: CBPeripheralManagerDelegate {

  func peripheralManagerDidUpdateState(_ peripheral: CBPeripheralManager) {
```

Bluetooth Connection and Detection

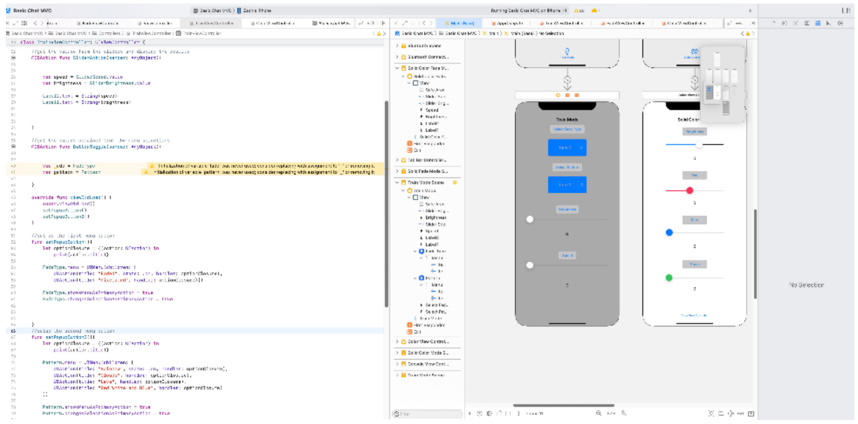
This is the code for detecting and connecting to a Bluetooth device if it is selected. Each function uses Core Bluetooth which is the swift function for all things related to Bluetooth. In the startScanning() function at the bottom is where I believe the issue is because of the CBUUID which uses a preset selection of UUID's which could have been for devices other than what I was trying to use which would cause it not to be detected.



```

    override func viewDidAppear(_ animated: Bool) {
      disconnectFromDevice()
      self.tableView.reloadData()
      //startScanning()
    }

    func connectToDevice() -> Void {
      centralManager?.connect(bluefruitPeripheral!, options: nil)
  }

    func disconnectFromDevice() -> Void {
      if bluefruitPeripheral != nil {
        centralManager?.cancelPeripheralConnection(bluefruitPeripheral!)
      }
  }

    func removeArrayData() -> Void {
      centralManager.cancelPeripheralConnection(bluefruitPeripheral)
           rssiArray.removeAll()
           peripheralArray.removeAll()
       }

    func startScanning() -> Void {
        // Remove prior data
        peripheralArray.removeAll()
        rssiArray.removeAll()
        // Start Scanning
        centralManager?.scanForPeripherals(withServices: [CBUUIDs.BLEService_UUID])
        scanningLabel.text = "Scanning..."
        scanningButton.isEnabled = false
        Timer.scheduledTimer(withTimeInterval: 15, repeats: false) {_ in
            self.stopScanning()
```

Buttons and Sliders

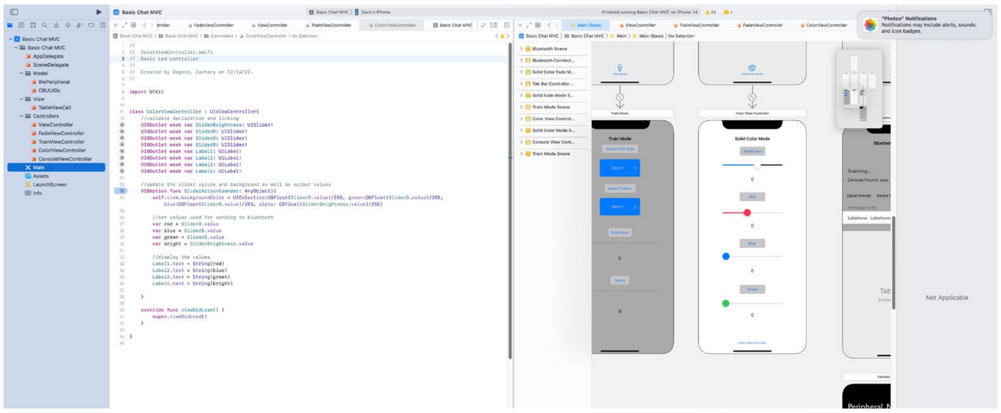
This shows the code for using inputs like buttons, sliders, and labels to show results of selection. After implementing in the Story Board on the right inputs/outputs must be linked with the code so that everything functions correctly. The lines of code starting with @IBOutlet are the declarations for each input/output, which are linked by connecting the small dots on the left side to the corresponding field in Story Board on the right. After this initialization the values can be read and write which happens right below in the SliderAction function which continually updates the values read in from the inputs as they change and outputs. This view is for the solid color mode which allows the user to select a color to be shown on the LEDS, to show the color it updates the background with the color based on the sliders which also occurs in the slider action function. Each of the other functions has varying inputs but generally follows the same format for inputs/outputs.



```
class TrainViewController: UIViewController {
    //get the values from the sliders and display the results
    @IBAction func SliderAction(sender: AnyObject){


        var speed = SliderSpeed.value
        var brightness = SliderBrightness.value

        Label2.text = String(speed)
        Label1.text = String(brightness)



    }

    //get the values obtained from the menu selections
    @IBAction func ButtonToggle(sender: AnyObject){


        var _fade = FadeType        ⚠ Initialization of variable '_fade' was never used; consider replacing with assignment to '_' or removing it
        var pattern = Pattern       ⚠ Initialization of variable 'pattern' was never used; consider replacing with assignment to '_' or removing it

    }

    override func viewDidLoad() {
        super.viewDidLoad()
        setPopupButton()
        setPopupButton2()
    }

    //set up the first menu button
    func setPopupButton(){
        let optionClosure = {(action: UIAction) in
            print(action.title)}

        FadeType.menu = UIMenu(children: [
            UIAction(title: "Faded", state: .on, handler: optionClosure),
            UIAction(title: "Pixelated", handler: optionClosure)])

        FadeType.showsMenuAsPrimaryAction = true
        FadeType.changesSelectionAsPrimaryAction = true



    }
    //setup the second menu button
    func setPopupButton2(){
        let optionClosure = {(action: UIAction) in
            print(action.title)}

        Pattern.menu = UIMenu(children: [
            UIAction(title: "Rainbow", state: .on, handler: optionClosure),
            UIAction(title: "Clouds", handler: optionClosure),
            UIAction(title: "Lava", handler: optionClosure),
            UIAction(title: "Red White and Blue", handler: optionClosure)
        ])

        Pattern.showsMenuAsPrimaryAction = true
        Pattern.changesSelectionAsPrimaryAction = true
```

Menu Button

This is the code for a drop-down menu button which has different options when you click on the drop down. Each drop-down button has a separate function for initialization and varies with the number of options but follow the same initial variable initiation and linking as the other buttons and sliders.



```
//
//  ColorViewController.swift
//  Basic Led controller
//
//  Created by Degore, Zachary on 12/14/22.
//

import UIKit

class ColorViewController : UIViewController{
    //variable declaration and linking
    @IBOutlet weak var SliderBrightness: UISlider!
    @IBOutlet weak var SliderR: UISlider!
    @IBOutlet weak var SliderG: UISlider!
    @IBOutlet weak var SliderB: UISlider!
    @IBOutlet weak var Label1: UILabel!
    @IBOutlet weak var Label2: UILabel!
    @IBOutlet weak var Label3: UILabel!
    @IBOutlet weak var Label4: UILabel!

    //update the slider values and background as well as output values
    @IBAction func SliderAction(sender: AnyObject){
        self.view.backgroundColor = UIColor(red:CGFloat(SliderR.value)/255, green:CGFloat(SliderG.value)/255,
            blue:CGFloat(SliderB.value)/255, alpha: CGFloat(SliderBrightness.value)/255)

        //set values used for sending to bluetooth
        var red = SliderR.value
        var blue = SliderB.value
        var green = SliderG.value
        var bright = SliderBrightness.value

        //display the values
        Label1.text = String(red)
        Label2.text = String(blue)
        Label3.text = String(green)
        Label4.text = String(bright)

    }

    override func viewDidLoad() {
        super.viewDidLoad()
    }

}
```

Arduino Code

The Arduino code is responsible for controlling the LED lights based on what is selected in the app and transmitted over Bluetooth. Most comments in the code explain what is happening but this will show the overview since the code can be viewed on a PC unlike the swift code which is only available on MAC.

Version 1.0

This code was very basic as it was just used attempting to get the LED display to function which ended up not being possible due to the constraints of the feather chip. It used the Arduino Protomatter library which can control a 64x64 display which uses 5 address bits instead of the typical 3 or 4 bits for smaller resolution displays. This code was set aside as it was not needed now that the project was going to be using LED strips.

Version 2.0

This code was used to now control LED strips, it took use of the FastLED library which has a variety of capabilities for many different types of LED strips. There are 4 main sections of the code, initialization, Bluetooth connection, reading and interpreting Bluetooth values, and LED control.

Initialization

This section holds the initialization for LED functions and Bluetooth. It begins by setting up Bluetooth variable bleuart which is used for Bluetooth communication based on the UUID, which is what caused the issue with the swift app. The FastLED variable is also set up using the led type, output pin, and LED order, this library.

Bluetooth

This section connects to devices if the chose to, then begins listening for transmission through bleuart. Once a transmission is detected it enters the loop and begins taking in values, however these values are received as ASCII values, so they need to be translated to regular decimal values. It also detects when a separating space or end of transmission has been sent to know when to append a value to the list or stop listening for information to then begin outputting to the LEDS.

LED control

Now that the values from transmission are stored in an array it can be separated into the values needed for LED control. These values are mode, red, green, blue, brightness, speed, smoothness, and pattern in that order. The first thing to do now is enter a switch which selects the mode based on the first value. Next, depending on the mode is to use all the other values to create an output, some functions only require a few values while others may require all these values. Once the output has been set, the switch is exited +-*and this process is repeated until new information is sent across Bluetooth. This allows the lights to continually output until new information is sent.

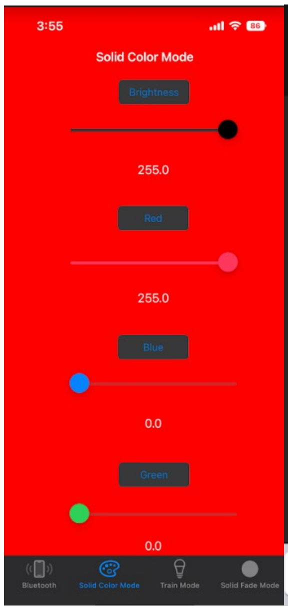
Git Hub Code link:

This link goes to the git hub repository with all of the code.

zdegore/juniordesignproject3: Design Project 3 for Junior Design (github.com)

Design Testing

Due to the Bluetooth connection not working with the app, testing had to be done through the Blue fruit Connect app which has basic features including a UART modes that allows you to send values to the feather chip. To demonstrate what would have happened I manually sent the values to the chip in the same format that they would have been in if it was through my app. The picture on the right below shows what the information sent looks like, as seen before it is(mode, R, G, B, brightness, speed, smoothness, and pattern). Picutre on the left shows what it would look like in my app. There is a very large number of options for between all of the mode so each mode will be shown with two different scenarios varying in options to keep it short.



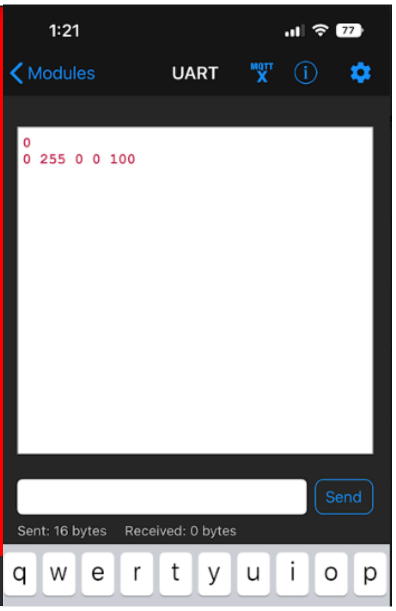
Solid Color Mode

Brightness 255.0

Red 255.0

Blue 0.0

Green 0.0

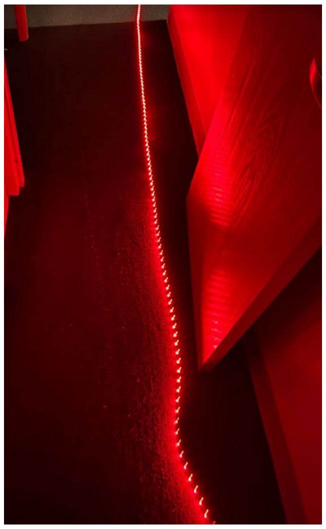


```
0
0 255 0 0 100
```

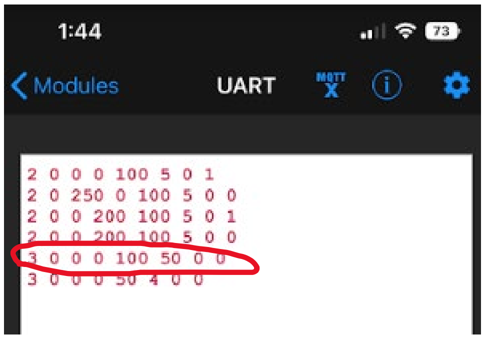
Sent: 16 bytes   Received: 0 bytes

In this case 0 255 0 0 100 was sent this corresponds to Mode: 0 (solid color)

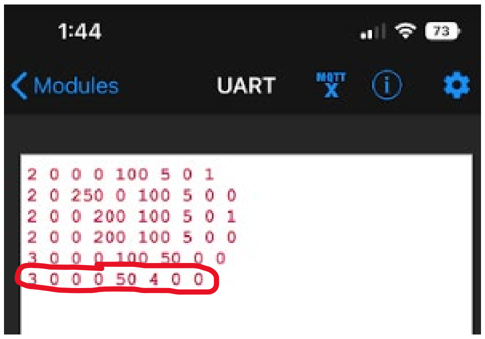
Color R(255) G(0) B(0) Brightness: (100)



This is the demonstration for the fade mode(2) and shows two levels of speed and brightness.
Left brightness(100) speed (50) Right brightness(50) speed(4)



```
2 0 0 0 100 5 0 1
2 0 250 0 100 5 0 0
2 0 0 200 100 5 0 1
2 0 0 200 100 5 0 0
3 0 0 0 100 50 0 0
3 0 0 0 50 4 0 0
```

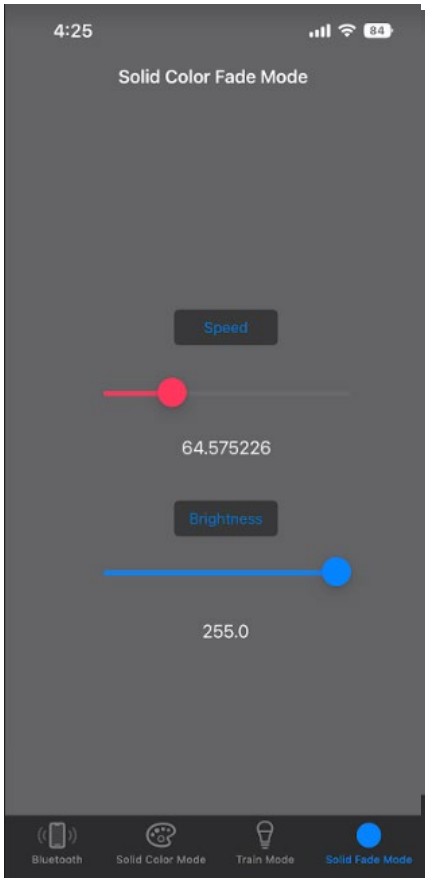


```
2 0 0 0 100 5 0 1
2 0 250 0 100 5 0 0
2 0 0 200 100 5 0 1
2 0 0 200 100 5 0 0
3 0 0 0 100 50 0 0
3 0 0 0 50 4 0 0
```

Double Click on videos to view.

IMG_9255.MOV

IMG_9259.MOV



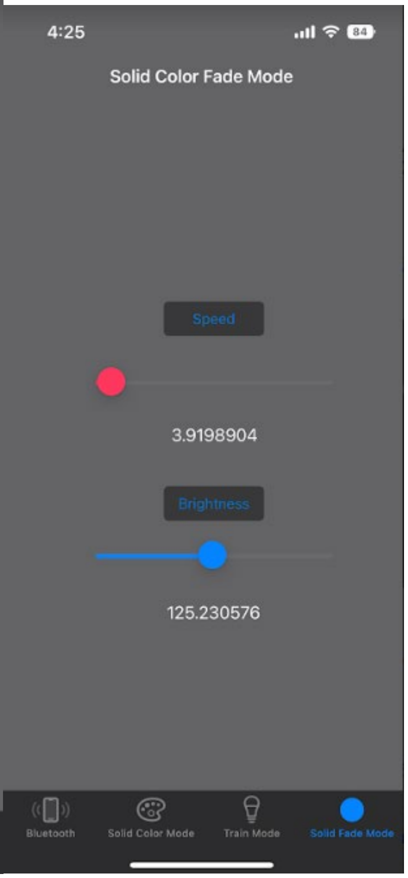
Solid Color Fade Mode

Speed

64.575226

Brightness

255.0



Solid Color Fade Mode

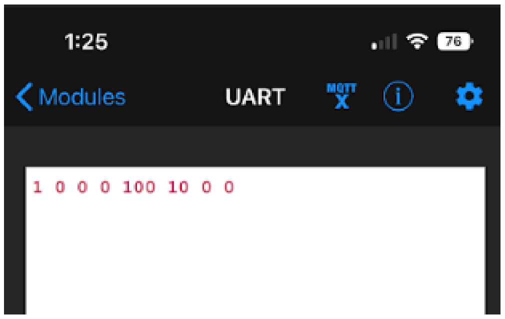
Speed

3.9198904

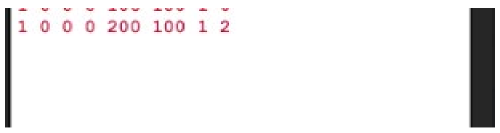
Brightness

125.230576

This demonstration is mode(1) color train



```
1 0 0 0 100 10 0 0
```

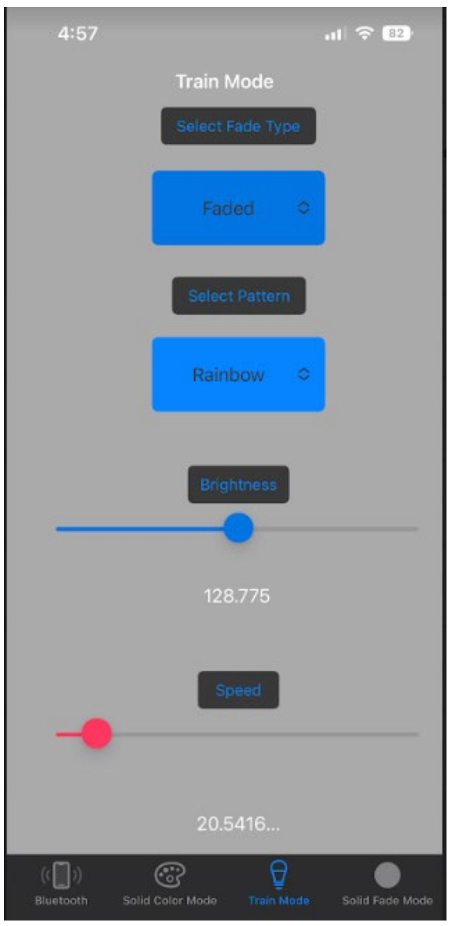


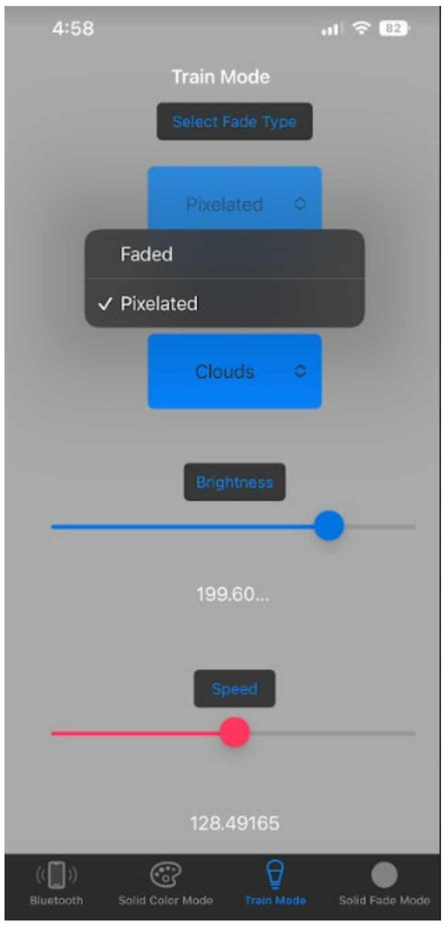
```
1 0 0 0 200 100 1 2
```

Left is mode(1) RGB(null) brightness(100) speed(10) smoothness(0/smooth) pattern(0/rainbow)

Left is mode(1) RGB(null) brightness(200) speed(100) smoothness(1/pixelated) pattern(2/clouds)

IMG_9249.MOV   IMG_9251.MOV





Summary

Despite the last-minute change to the project and Bluetooth functionality not working correctly, I was satisfied with what I had learned completing this project. I started with no swift coding experience, and I believe that I learned a fair amount from what worked and from what didn’t work. I also strengthened my coding knowledge in Arduino using the various libraries. Looking back the first thing I would change would to have made sure 100% that all the parts worked together on the software and hardware side. This would have prevented the issues I faced and would likely have allowed me to have a fully functional final project. I also would have come up with a better secondary plan as I spent too much time trying to get the original design to work before attempting to try something else. I really would have liked to have the original design finished and functional, but I believe I learned to better myself for future projects through my errors in this project.