

27 de Junho de 2015

Primeiro dia de organização das tarefas

To Do, Doing, Done

Como funciona

Bom, antes de começar qualquer projeto gosto de planejar tudo o que tenho que fazer, pois assim consigo administrar tudo o que tenho que fazer.

Este método funciona da seguinte maneira, eu divido em três partes o desenvolvimento do projeto, em coisas que precisam ser feitas (To Do), em coisas estão sendo feitas e em coisas já terminadas.

Utilizei post-it para ter uma melhor visualização de um contexto geral do projeto e conseguir medir o que falta e o que não falta a ser feito.

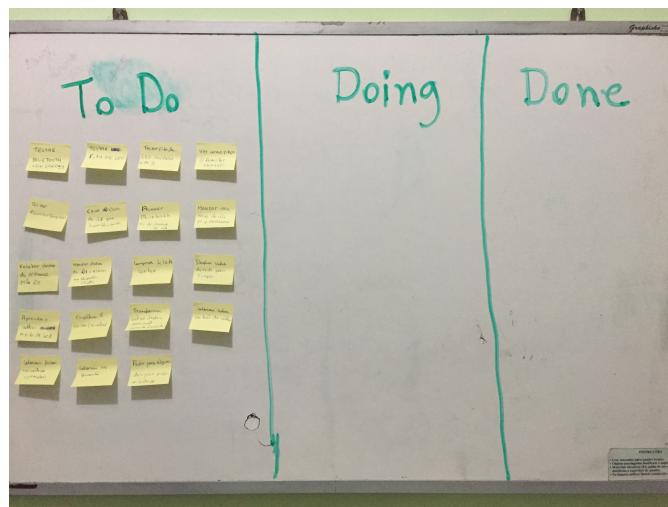


Figure 1: Lousa com Post-its

Bluetooth Low Energy 4.0

Testando Bluetooth Low Energy no Arduino

Foi feito um circuito simples que ligasse o Bluetooth Low Energy, chamado também de BLE, que fosse testado se era possível achar pelo iOS o BLE ligado ao Arduino.

Foi percebido (depois de um longo tempo tentando identificar o problema) que no menu do Bluetooth do iOS é impossível achar Bluetooth's que não são dos dispositivos da Apple. Portanto, é necessário baixar um aplicativo externo para testá-lo, no meu caso eu fiz o download do LightBlue, disponível na AppStore para iPhone.

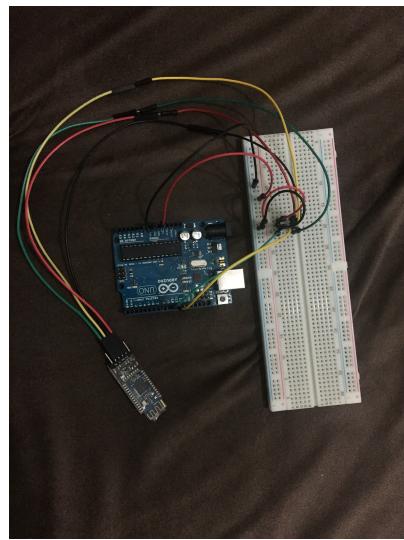


Figure 2: Teste com BLE no Arduino

28 de Junho de 2015

Segundo dia de organização das tarefas

Criando código para acessar o BLE

Em Swift

Bom, eu procurei diversos tutoriais para iOS para se conectar com o BLE 4.0, mas em Swift eu só achei um projeto chamado "BleModuleMaster" e a partir dele, eu comentei o código e futuramente pretendo criar uma biblioteca para que qualquer um possa utilizar.

O Aplicativo hoje só tem apenas um botão e uma tableView, que mostra todos os dispositivos próximos. Quando eu clico em um destes dispositivos ele conecta com o Bluetooth que eu indiquei.

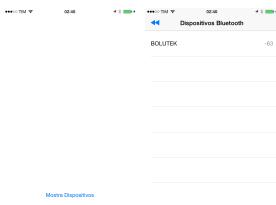


Figure 3: Primeiras imagens do Aplicativo

A parte mais complicada de hoje foi tentar enviar os dados do iOS para o Arduino, eu pensava que existia uma grande conversão entre bites e bytes e pensava que seria mais complicado que eu pensava. O algoritmo na ordem de execução fica desta forma:

- iOS - Primeiro a função recebe a String
- iOS - Depois ela converte essa String em formato de NSData(data eu digo em formato de buffers)
- iOS - Mando os dados para o Arduino
- Arduino - Lê caractere por caractere
- Arduino - Retorna pro iOS o mesmo caractere enviado
- iOS - Receberá o caractere enviado

```
#include <SoftwareSerial.h>

SoftwareSerial mySerial(3,4); // RX, TX
char a;
void setup() {
    //Declara os seriais
    Serial.begin(9600);
    mySerial.begin(9600);

    delay(1000);

    Serial.println("Setup done");
}

void loop() {
    // Verifica se recebeu algo
    if (mySerial.available()){
        Serial.print("Received data: ");

        //Enquanto tem caracteres ele continua lendo
        while (mySerial.available()) {
            a = (char) mySerial.read();

            Serial.print(a);
            if (a != NULL)
            {
                mySerial.write(a); // loop back iOS
            }
        }
        Serial.println("");
        delay(1000);
    }

    delay(1000);
}

//Escreve um valor novo no BLE
func.writeValue(data: String){
    let data = (data as NSString).dataUsingEncoding(NSUTF8StringEncoding)

    print("Escreve no arduino1")
    if let peripheralDevice = peripheralDevice{
        print("Escreve no arduino3")
        if let deviceCharacteristics = deviceCharacteristics {

            //Pega a String e manda para o arduino
            peripheralDevice.writeValue(data!, forCharacteristic: deviceCharacteristics, type: CBCharacteristicWriteType.WithResponse)
            print("Escreve no arduino")
        }
    }
}

// Quando é recebido algo do Device - Retorno
func peripheral(peripheral: CBPeripheral?, didUpdateValueForCharacteristic characteristic: CBCharacteristic?, error: NSError!) {

    let ret = NSString(data:characteristic!.value!, encoding: NSUTF8StringEncoding) as! String
    println("Retorno - \(ret)")
}
```

Figure 4: Imagens do código mostrando os primeiros exemplos de leitura e captura de dados via Bluetooth com Swift

2 de Julho de 2015

Planejamento de telas, aprendendo register shifter,

Planenjamento de telas

Antes de começar a falar das telas, eu queria dizer que nos ultimos dias eu não trabalhei pelo fato de estar ainda no final do semestre, e tive que entregar alguns trabalhos por isso fiquei estes três dias sem desenvolver, mas estava pensando por um thread da minha cabeça.

Hoje antes de jogar paintball, enquanto eu estava no metrô eu planejei o desenho mais ou menos das telas do aplicativo, vou colocar aqui meu esboço.

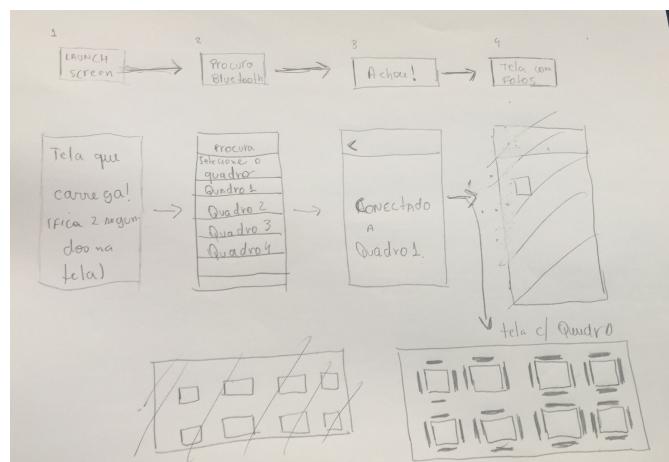


Figure 5: Fluxo de tela muito ruim que eu fiz no Metrô

Register Shifter

Convertendo decimal para binário

E enquanto eu voltava do paintball, eu planejei como eu usaria o register shifter e vi que ele recebe um número decimal em binário, e a partir destes a saída os pinos de saída são liberados.

Com isso para acender os oito quadros, eu tive que pensar, como posso fazer o shifter acender um quadro, colocando a combinação de binários do mesmo. Exemplo: Para acender o primeiro, eu preciso da sequência 10000000, que no caso é o numero 128. O segundo quadro é 01000000, que é 64 e assim por diante em todos os quadros.

O que ocorre é o que acontece quando eu quiser acender mais que um quadro de uma vez? Então, é necessário somar os números decimais e dos respectivos quadros, e depois convertê-los para binários, assim, o binário que surgirá, terá os dois quadros.

Exemplo: Se eu quero quadro 1 e o quadro 5.

Quadro - Decimal

1 - 128 —— 2 - 64 —— 3 - 32 —— 4 - 16 —— 5 - 8 —— 6 - 4 —— 7 - 2 —— 8 - 1

Logo eu somaria 128 + 8, que daria 136. E 136 em binário seria 10001000 e assim acenderia todos os quadros necessários que fossem pedidos. Essa parte de soma seria pelo iOS e a conversão pelo Arduino. Bom, amanhã colocarei essa teoria em prática.

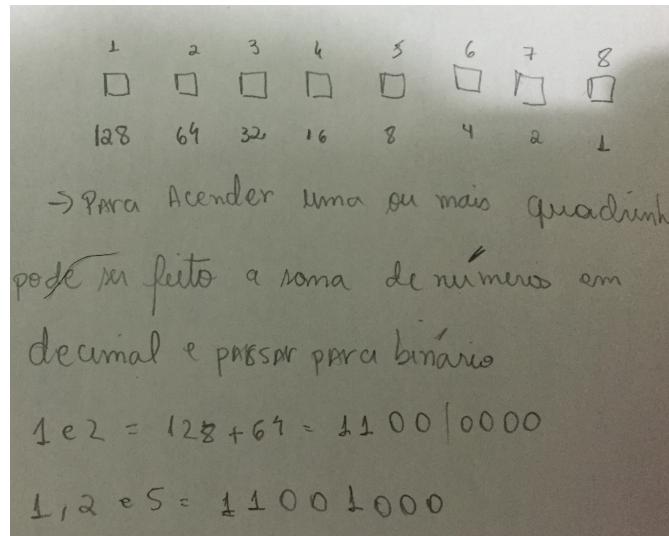


Figure 6: Conversão de decimal para binários com quadros