Politécnico de Viseu
Tecnologia e Gestão Lamego

Trabalho prático Sistemas de Telecomunicações

Trabalho de avaliação à disciplina de Sistemas de Telecomunicações

Politécnico de Viseu
Tecnologia e Gestão Lamego

LAMEGO 2024

Projeto Montain Wolves Airsoft Bomb

Trabalho de avaliação à disciplina de Sistemas de Telecomunicações

Licenciatura: Engenharia Informática e Telecomunicações

Cadeira: Sistemas de Telecomunicações

Orientador: Prof. Jorge Duarte

Autor: Jorge Pinto nº26171

Introdução

Como estudante universitário e praticante de Airsoft na equipa Montain Wolves, decidi criar um projeto pessoal que pudesse ser integrado na disciplina e que cumprisse todos os requisitos necessários. Este projeto consiste em uma “Fake Bomb” para ser utilizada em ambientes de jogo de Airsoft.

O projeto possui múltiplas possibilidades de atualização, tais como:

- Operar como um sistema simples de efeito de bomba,
- Integrar-se a um protocolo de comunicação para interação com outros dispositivos durante o jogo,
- Conectar-se a um servidor, permitindo, através de sensores de localização, o monitoramento em tempo real de sua posição, criando assim uma espécie de "streaming" do jogo.

O dispositivo irá apresentar as seguintes funcionalidades:

- Configuração via Bluetooth,
- Configuração manual,
- Localização do dispositivo,

Politécnico de Viseu
Tecnologia e Gestão Lamego

Índice

Tema do projeto

Para a construção física do projeto, foi utilizado como base o modelo descrito a seguir. Os links presentes na bibliografia apresentam todos os componentes do projeto original, incluindo peças 3D e código. O código do projeto original é amplamente utilizado em diversos projetos semelhantes no universo do Airsoft. No entanto, a ideia central deste trabalho foi reutilizar a estrutura física com o propósito de adaptá-la às nossas necessidades específicas. Assim, a estrutura serviu apenas como um ponto de partida e precisou ser redesenhada. Para esse processo, foi utilizado o software de modelagem 3D SolidWorks.

Figura 1 - Protejo Inicial

Linguagem de programação

No desenvolvimento do projeto, foram empregues diferentes linguagens de programação, de acordo com as necessidades específicas de cada ambiente. O C++ foi utilizado para a programação do microcontrolador, garantindo eficiência e controle detalhado dos recursos embarcados. Já a linguagem Java foi aplicada no desenvolvimento do aplicativo Android, proporcionando uma interface amigável e uma integração robusta com o sistema móvel.

C++

C++ é uma linguagem de programação de propósito geral conhecida por ser poderosa e versátil, derivada do C. Ela oferece suporte à programação orientada a objetos, programação genérica e programação procedural. Seu principal diferencial é a combinação de eficiência de baixo nível (próxima ao hardware) com recursos de alto nível, o que a torna ideal para sistemas de software complexos, como sistemas operacionais, drivers de dispositivos, aplicações em tempo real, jogos e aplicações de alta performance.

C++ oferece suporte a encapsulamento, herança e polimorfismo, permitindo modularidade e reuso de código, além de controle detalhado de recursos, como memória, por meio de ponteiros e gerenciamento manual. Sua flexibilidade também se estende ao desenvolvimento de sistemas embarcados, como o ESP32, onde é amplamente utilizado para criar aplicações eficientes e com recursos avançados, tirando proveito das capacidades de conectividade Wi-Fi e Bluetooth do microcontrolador.

Java

Java é uma linguagem de programação orientada a objetos, conhecida por sua portabilidade, robustez e facilidade de uso. Desenvolvida para ser independente de plataforma através da Máquina Virtual Java (JVM), ela é amplamente utilizada no desenvolvimento de aplicativos, especialmente para dispositivos Android.

No contexto do projeto, Java está sendo utilizado para o desenvolvimento de um aplicativo Android. Essa escolha permite a criação de interfaces interativas e a integração de funcionalidades específicas para o controle e monitoramento do dispositivo desenvolvido. A flexibilidade da linguagem, aliada a um vasto ecossistema de bibliotecas e ferramentas, facilita o desenvolvimento de aplicações móveis robustas e escaláveis, garantindo uma experiência de utilizador fluida.

Microcontroladores

Os microcontroladores desempenham o papel de "cérebro" neste projeto, pois são responsáveis, por meio da programação em C++, pelo controle da lógica do sistema. Eles

permitem a leitura de sinais de entrada e a gestão das saídas, viabilizando o funcionamento desejado do dispositivo. Atualmente, há diversos tipos de microcontroladores disponíveis no mercado, dos quais alguns dos principais serão mencionados mais adiante neste trabalho.

Softwares utilizados

Arduino IDE

O Arduino IDE (Integrated Development Environment) é um ambiente de desenvolvimento integrado utilizado para programar placas Arduino e outros microcontroladores compatíveis. Ele oferece uma interface simples e acessível, projetada para facilitar a programação e a prototipagem de dispositivos eletrônicos, mesmo para iniciantes. O Arduino IDE suporta a linguagem de programação baseada em C/C++, tornando o desenvolvimento prático e intuitivo por meio de funções predefinidas e bibliotecas amplamente utilizadas pela comunidade.

Com o Arduino IDE, é possível escrever, compilar e carregar código para microcontroladores através de uma conexão USB. A ferramenta oferece recursos como detecção automática de portas seriais, um monitor serial para depuração em tempo real, além de um vasto repositório de exemplos e bibliotecas que ajudam a acelerar o desenvolvimento de projetos. Sua versatilidade e compatibilidade com uma ampla gama de placas, incluindo módulos baseados no ESP32, tornam o Arduino IDE uma escolha popular para prototipagem de sistemas embarcados, projetos IoT e automação em geral.

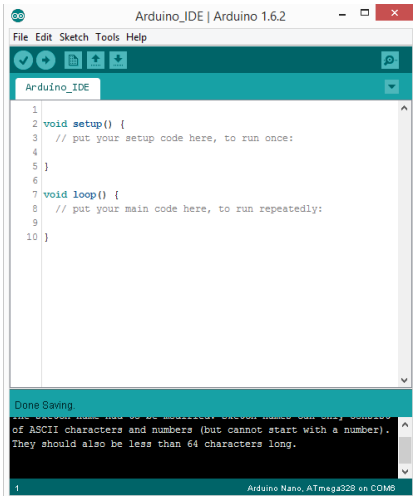


Figura 2 - Arduino IDE

SolidWorks

Como o nosso projeto irá ter muitas mais funções que o projeto original algumas peças tiveram de ser redesenhadas, nomeadamente a tampa. A tampa em vez de agregar botões como o projeto inicial a mesma irá agregar os seguintes componentes:

- Antena GPS e placa de controlo
- Display lcd Oled inteiramente dedicado ao GPS
- Buzzer

Android Studio

O Android Studio é o ambiente de desenvolvimento integrado (IDE) oficial para o desenvolvimento de aplicativos Android. Baseado no IntelliJ IDEA, o Android Studio foi projetado pela Google para oferecer uma experiência de desenvolvimento poderosa e flexível, fornecendo todas as ferramentas necessárias para criar, depurar e testar aplicativos móveis para dispositivos Android.

O Android Studio oferece um editor de código inteligente com sugestões e autocompletar, suporte integrado ao desenvolvimento com a linguagem Java, Kotlin e C++, além de ferramentas gráficas de design para construir interfaces de utilizador de forma visual. Possui também emuladores para testar aplicativos em uma ampla gama de dispositivos e configurações, bem como um sistema de compilação baseado no Gradle, que facilita a automação de tarefas e gerência de dependências.

O ambiente ainda traz ferramentas para otimização de desempenho, depuração de código, testes automatizados, suporte a desenvolvimento de aplicações para diferentes tamanhos de tela e dispositivos, além de integração com o Firebase para fornecer funcionalidades como autenticação, armazenamento de dados em tempo real, entre outros. Tudo isso faz do Android Studio a solução mais robusta e completa para o desenvolvimento de aplicativos Android.

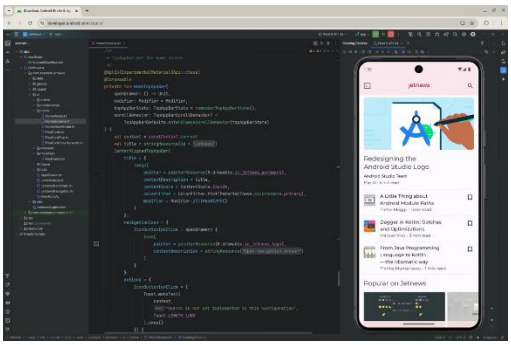


Figura 3 - Android Studio

Git

Git é um sistema de controle de versão distribuído amplamente utilizado no desenvolvimento de software. Ele permite que desenvolvedores rastreiem alterações em seu código-fonte, colaborem de maneira eficiente em equipes e revertam para versões anteriores do projeto quando necessário. Criado por Linus Torvalds em 2005, o Git é conhecido por sua rapidez, confiabilidade e flexibilidade.

Com Git, as mudanças são registradas em um repositório que armazena todo o histórico de modificações, possibilitando o controle detalhado das versões do código. Cada desenvolvedor possui uma cópia local do repositório, permitindo o trabalho offline e a realização de experimentos sem impactar a versão principal do projeto. Ele utiliza conceitos como branches (ramificações) para que múltiplos desenvolvedores possam trabalhar em diferentes funcionalidades ou correções de forma isolada antes de mesclá-las ao projeto principal.

Além disso, Git facilita a colaboração por meio de plataformas de hospedagem como GitHub, GitLab e Bitbucket, que oferecem recursos adicionais como gerenciamento de issues, revisões de código, e integração contínua (CI/CD). Essas funcionalidades tornam Git uma ferramenta essencial para equipes de desenvolvimento modernas, ajudando a gerenciar o ciclo de vida dos projetos de forma eficiente e segura.

Figura 4 - Git

Componentes Utilizados no Projeto

Esp32 Dev Kit V4

O ESP32 é um microcontrolador altamente versátil desenvolvido pela Espressif Systems, conhecido por suas capacidades integradas de Wi-Fi e Bluetooth. Ele é amplamente utilizado em projetos de Internet das Coisas (IoT), automação, robótica e dispositivos embarcados devido à sua alta performance e custo acessível.

As suas principais caracteristicas:

- Conectividade: Possui suporte a Wi-Fi 2.4 GHz (padrão 802.11 b/g/n) e Bluetooth (Classic e BLE).
- Alta Performance: Equipado com um ou dois núcleos Tensilica Xtensa LX6, rodando até 240 MHz, com memória RAM (SRAM) integrada.
- Portas de Entrada/Saída: Conta com GPIOs versáteis, suportando PWM, ADC, DAC, I2C, SPI, UART, entre outros protocolos.
- Baixo Consumo de Energia: Oferece modos de economia de energia, como Deep Sleep, para aplicações que exigem eficiência energética.
- Flexibilidade: Compatível com várias plataformas de desenvolvimento, como Arduino IDE, PlatformIO, ESP-IDF (SDK oficial da Espressif) e MicroPython.
- Armazenamento: Geralmente inclui memória flash integrada para armazenar firmware e dados.
- O ESP32 é ideal para projetos como sensores inteligentes, dispositivos de controle remoto, redes mesh e qualquer aplicação que exija conectividade sem fio com processamento local eficiente.

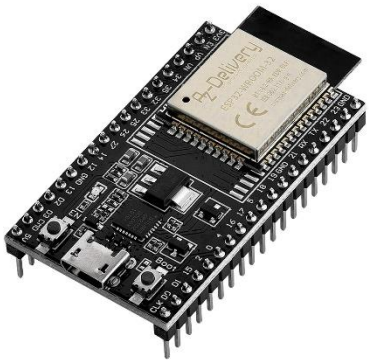


Figura 5 - ESP 32

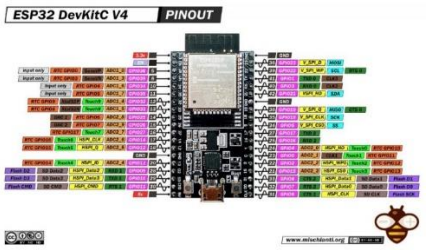


Figura 6 - ESP 32 Pinout

Protoboard

Uma protoboard, também conhecida como breadboard, é uma placa reutilizável usada para criar circuitos eletrônicos de forma prática e sem a necessidade de solda. Ela é amplamente utilizada por estudantes, hobbistas e engenheiros para testes, experimentos e desenvolvimento de protótipos.

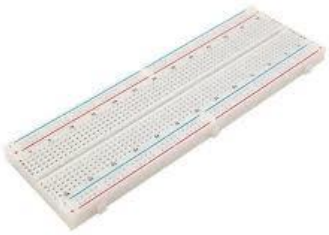


Figura 7 - Protoboard

Placa PCB perfurada

Uma placa PCB perfurada, ou placa de fenolite perfurada, é um tipo de placa de circuito impresso (PCB) genérica, projetada para montagem manual de circuitos eletrônicos. Ao contrário das PCBs convencionais, que possuem trilhas condutoras já impressas, a placa perfurada é composta apenas por uma matriz de furos regularmente espaçados, permitindo total liberdade no layout do circuito.

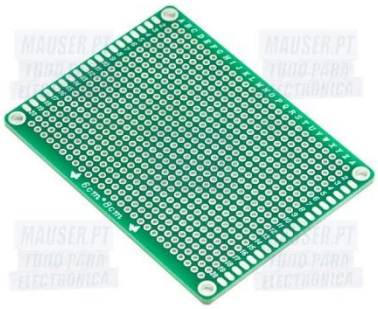


Figura 8 - PCB perfurada

Display OLED

Um display OLED (Organic Light-Emitting Diode) é um tipo de tela que utiliza diodos orgânicos emissores de luz para produzir imagens. Ele é amplamente utilizado em

dispositivos eletrônicos devido à sua qualidade visual superior, baixo consumo de energia e design compacto.

Figura 9 - Display Oled

Display Liquid Crystal I2C

Um display de cristal líquido (LCD) com interface I2C é uma variação dos tradicionais displays LCD que utiliza um módulo controlador I2C para facilitar a comunicação com microcontroladores. Essa configuração reduz a quantidade de conexões necessárias, tornando o uso mais prático em projetos eletrônicos.

Figura 10 - Display Liquid Crystal

Módulo GPS

Um módulo GPS é um dispositivo eletrônico que permite determinar a posição geográfica, velocidade e tempo exato, utilizando sinais de satélites do Sistema de

Posicionamento Global (GPS). Ele é amplamente usado em aplicações de navegação, rastreamento e localização em tempo real.

Figura 11 - Módulo GPS

Teclado Matricial 4x4

Um teclado matricial é um tipo de teclado projetado em uma matriz de linhas e colunas para reduzir o número de conexões necessárias entre os botões e o controlador (como um microcontrolador). Ele é amplamente utilizado em projetos eletrônicos para fornecer entradas simples e eficientes, como em interfaces de utilizador e sistemas de controle.

Figura 12 - Teclado matrivcial

Buzzer

Um buzzer é um componente eletrônico que emite som quando energizado. Ele é amplamente utilizado para fornecer alertas sonoros, sinais ou feedback em dispositivos eletrônicos.

Figura 13 - Buzzer

Leds WS2812b

Os LEDs WS2812B são dispositivos inteligentes que integram um LED RGB e um controlador em um único encapsulamento. Eles são amplamente utilizados em

projetos de iluminação decorativa, sinalização e displays personalizados devido à sua facilidade de controle e flexibilidade.

Estes leds possuem características diferentes dos convencionais, características estas que são:

- Controle Digital: Cada LED possui um controlador integrado que permite ajustar individualmente as cores e o brilho por meio de um protocolo de comunicação serial.
- Protocolo de Dados: Utiliza um único pino para transmissão de dados (linha de dados), simplificando a fiação. Os dados são enviados em cascata, ou seja, um LED retransmite os dados para o próximo na cadeia.
- Alimentação: Operam com uma tensão de 5V, contudo alguns modelos apenas toleram 3,3V no pino de dados.
- Alto Brilho e Cor Personalizável: Oferecem uma ampla gama de cores (24 bits por pixel) e podem ser configurados para animar padrões e transições de maneira fluida.
- Encadeamento: É possível conectar centenas de LEDs em série, desde que o fornecimento de energia seja adequado.
- Os WS2812B são populares em aplicações como fitas de LED endereçáveis, painéis de LED e iluminação de efeitos em projetos de eletrônica e design.

Figura 14 - Leds WS2812b

Esquemas e layouts

Esquema eletrônico

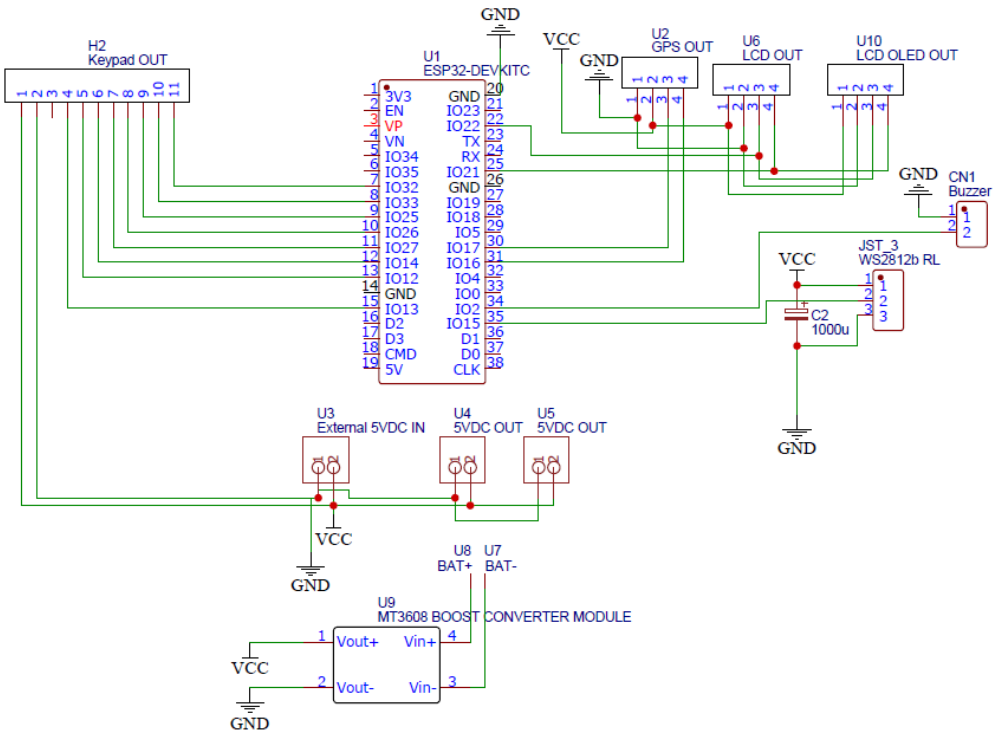


Figura 15 - Esquema eletrônico

Layout para PCB

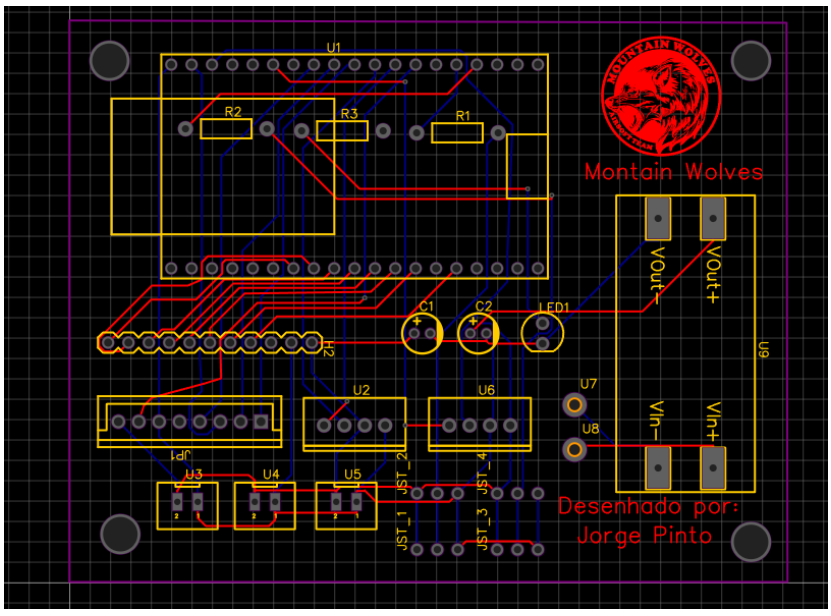


Figura 16 - Layout 2D da PCB

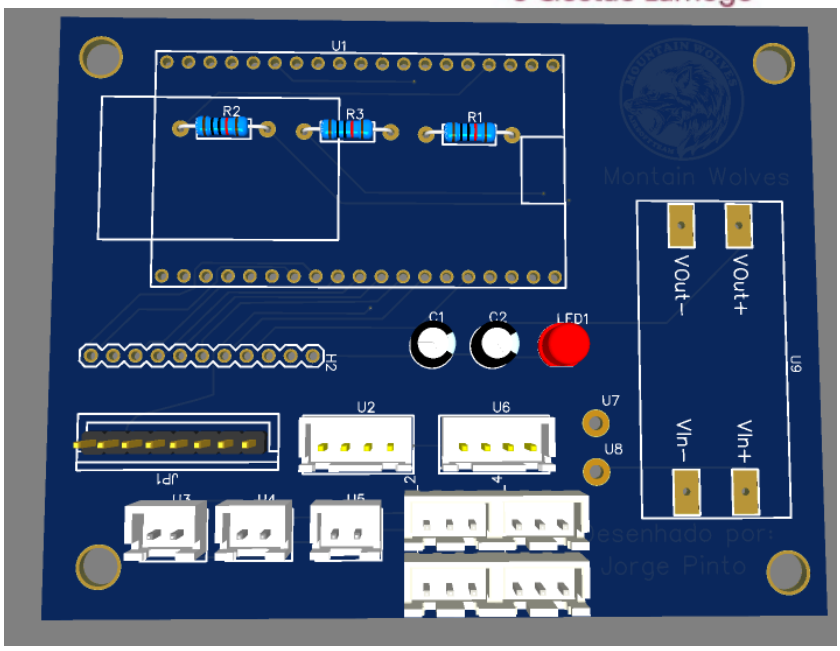


Figura 17 - Layout 3D da PCB (vista de Cima)

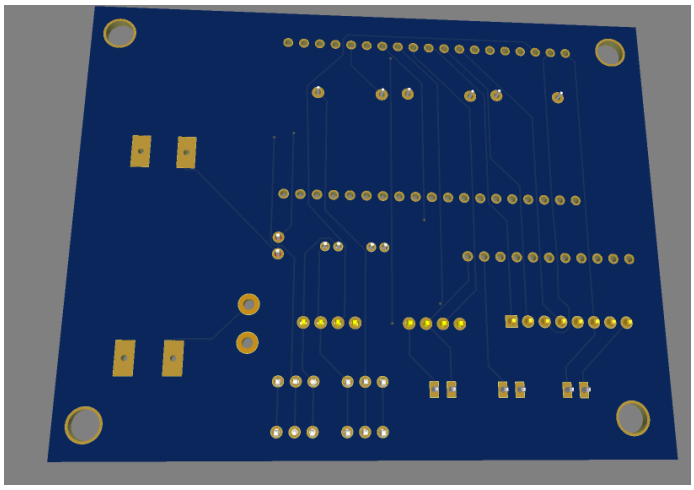
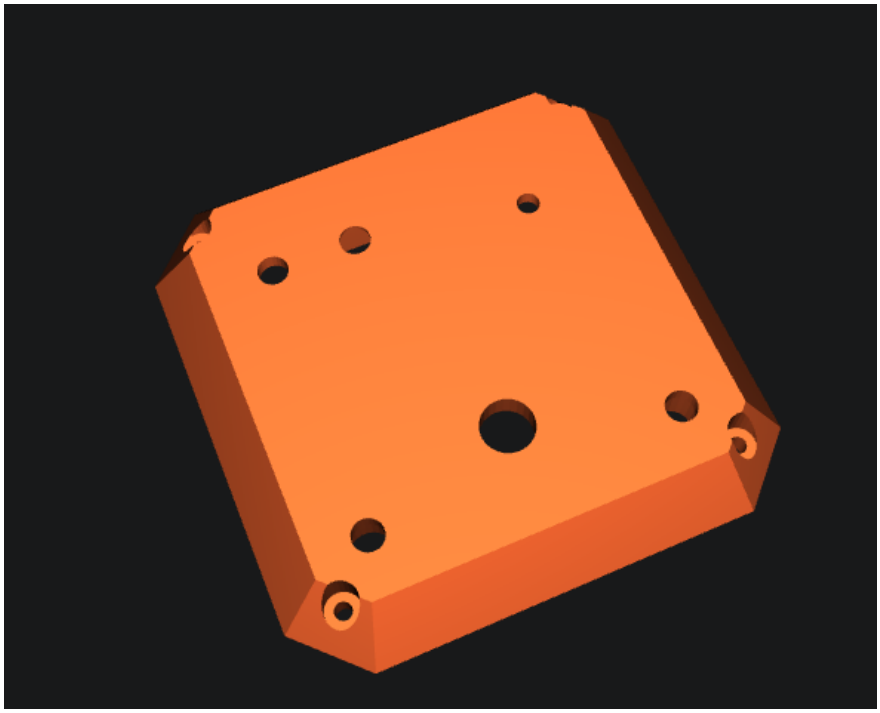


Figura 18 - Layout 3D da PCB (vista por baixo)

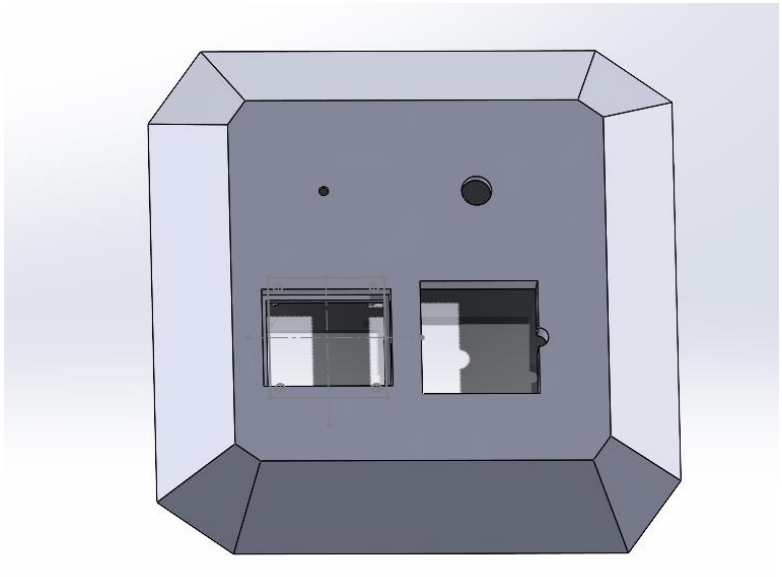
Modelo 3D

Tampa

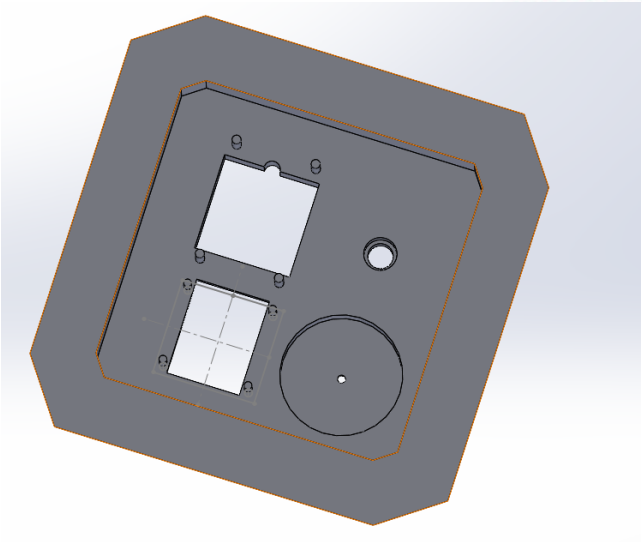
Original



Novo modelo



Politécnico de Viseu
Tecnologia e Gestão Lamego



Aplicação Android

Estrutura e design

A aplicação foi desenvolvida em Android Studio, logo o seu design é composto por ficheiros XML que neste caso vão ser 3:

- Activity Main: o ficheiro principal da “view” da aplicação
- Colors: onde será declarado o código de cores
- Strings: onde será declarado as strings a usar

Activity Main xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">
```

```
    <ImageView
        android:id="@+id/teamLogo"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:background="@drawable/montain_wolves_logo"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.0"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.0" />

    <ImageButton
        android:id="@+id/play_pause_intro"
        android:layout_width="24dp"
        android:layout_height="24dp"
        android:layout_marginTop="5dp"
        android:layout_marginEnd="5dp"
        android:background="@null"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:srcCompat="@drawable/ic_pause" />

    <TextView
        android:id="@+id/editText_inOut"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_marginStart="90dp"
        android:text="@string/in_out_layout"
        android:textColor="@color/green"
        android:textSize="30dp"
        android:textStyle="bold"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />


    <Switch
        android:id="@+id/sw_gps"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="10dp"
        android:layout_marginTop="50dp"
        android:background="@color/white"
        android:text="@string/switch_gps_off"
        android:textSize="25dp"
        android:textStyle="bold"
        android:textColor="@color/blue"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <Switch
        android:id="@+id/sw_sound"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="36dp"
        android:background="@color/white"
        android:text="@string/switch_sound_off"
        android:textSize="25dp"
        android:textStyle="bold"
        android:textColor="@color/blue"
```

```
        app:layout_constraintBottom_toBottomOf="@+id/sw_gps"
        app:layout_constraintStart_toEndOf="@+id/sw_gps"
        app:layout_constraintTop_toTopOf="@+id/sw_gps"
        app:layout_constraintVertical_bias="0.0" />

    <Switch
        android:id="@+id/sw_leds"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="20dp"
        android:text="@string/switch_leds_off"
        android:background="@color/white"
        android:textSize="25dp"
        android:textColor="@color/blue"
        android:textStyle="bold"
        app:layout_constraintEnd_toEndOf="@+id/sw_gps"
        app:layout_constraintStart_toStartOf="@+id/sw_gps"
        app:layout_constraintTop_toBottomOf="@+id/sw_gps" />

    <Switch
        android:id="@+id/sw_smoke"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="25dp"
        android:textColor="@color/blue"
        android:background="@color/white"
        android:text="@string/switch_smoke_off"
        android:textSize="25dp"
        android:textStyle="bold"
        app:layout_constraintEnd_toEndOf="@+id/sw_sound"
        app:layout_constraintStart_toStartOf="@+id/sw_sound"
        app:layout_constraintTop_toBottomOf="@+id/sw_sound" />

    <TextView
        android:id="@+id/editText_dataGame"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginLeft="128dp"
        android:layout_marginTop="20dp"
        android:text="@string/game_data_layout"
        android:textColor="@color/green"
        android:textSize="30dp"
        android:textStyle="bold"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintTop_toBottomOf="@id/sw_smoke" />

    <EditText
        android:id="@+id/edit_text_time_game"
        android:layout_width="200dp"
        android:layout_height="50dp"
        android:layout_marginStart="30dp"
        android:layout_marginTop="30dp"
        android:background="@color/white"
        android:ems="10"
        android:hint="Time Game"
        android:inputType="number"
        android:textAlignment="center"
        android:textSize="30dp"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/editText_dataGame"
```

```
/>

    <EditText
        android:id="@+id/edit_text_time_bomb"
        android:layout_width="200dp"
        android:layout_height="50dp"
        android:layout_marginStart="28dp"
        android:layout_marginTop="28dp"
        android:background="@color/white"
        android:ems="10"
        android:hint="Bomb Time"
        android:inputType="number"
        android:textAlignment="center"
        android:textSize="30dp"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/edit_text_time_game"
/>

    <Button
        android:id="@+id/btn_send_config"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="28dp"
        android:layout_marginTop="108dp"
        android:background="@color/red"
        android:text="@string/send_config"
        android:textSize="30dp"
        android:textStyle="bold"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/edit_text_bomb_Code"
/>

    <EditText
        android:id="@+id/edit_text_players"
        android:layout_width="150dp"
        android:layout_height="50dp"
        android:layout_marginStart="28dp"
        android:layout_marginTop="120dp"
        android:background="@color/white"
        android:ems="10"
        android:hint="@string/num_of_players"
        android:inputType="number"
        android:textAlignment="center"
        android:textSize="30dp"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/edit_text_time_game"
/>

    <EditText
        android:id="@+id/edit_text_bomb_Code"
        android:layout_width="200dp"
        android:layout_height="50dp"
        android:layout_marginStart="28dp"
        android:layout_marginTop="32dp"
        android:background="@color/white"
        android:ems="10"
        android:hint="@string/bomb_code"
        android:inputType="number"
        android:textAlignment="center"
        android:textSize="30dp"
```

```
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/edit_text_players"
/>

    <Button
        android:id="@+id/setCoordinates"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="28dp"
        android:layout_marginTop="12dp"
        android:background="@color/green"
        android:text="Set Coordinates"
        android:textSize="20dp"
        android:textStyle="bold"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/edit_text_bomb_Code"
/>
</androidx.constraintlayout.widget.ConstraintLayout>
```

Colors xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="black">#FF000000</color>
    <color name="white">#FFFFFFFF</color>
    <color name="green">#8BC34A</color>
    <color name="red">#F44336</color>
    <color name="blue">#0027FF</color>
</resources>
```

Strings xml

```
<resources>
    <string name="app_name">Montain Wolves</string>
    <string name="in_out_layout">Inputs / Outputs</string>
    <string name="game_data_layout">Data Game</string>
    <string name="switch_gps_off">GPS OFF</string>
    <string name="switch_gps_on">GPS ON</string>
    <string name="switch_leds_off">Leds OFF</string>
    <string name="switch_leds_on">Leds ON</string>
    <string name="switch_sound_off">Sound OFF</string>
    <string name="switch_sound_on">Sound ON</string>
    <string name="switch_smoke_off">Smoke OFF</string>
    <string name="switch_smoke_on">Smoke ON</string>
    <string name="time_of_game">Time</string>
    <string name="num_of_players">Players</string>
    <string name="bomb_code">Bomb Code</string>
    <string name="send_config">Done</string>
</resources>
```

Código Java

Os arquivos de design em XML serão integrados ao Back-end escrito em Java através da criação e declaração de objetos que representam os elementos da interface de utilizador (Views), como botões, caixas de texto, entre outros. Além disso, serão criados objetos para os componentes do sistema, como o módulo Bluetooth.

Após as suas declarações o sistema irá funcionar através de:

- Callbacks e Eventos: O sistema operará com callbacks, permitindo que cada elemento execute eventos de forma independente, independentemente de sua posição na interface. Esses eventos serão configurados com Listeners, como OnClickListener.
- Envio de Mensagens via Bluetooth: Quando um evento for disparado, será gerada uma mensagem correspondente. Essa mensagem será enviada diretamente ao dispositivo Bluetooth do ESP32, utilizando o endereço MAC 08:A6:F7:20:B3:1E. A comunicação será feita por meio da API Bluetooth do Android (BluetoothSocket e OutputStream), garantindo a entrega da mensagem ao ESP32.

As mensagens enviadas serão formatadas de forma que o ESP32 possa interpretá-las e executar as ações necessárias no projeto.

```
package com.example.montainwolvesapp;
import static android.content.ContentValues.TAG;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.bluetooth.BluetoothSocket;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.content.pm.PackageManager;
import android.media.MediaPlayer;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.ImageButton;
import android.widget.ImageView;
import android.widget.Switch;
import android.widget.Toast;
import androidx.appcompat.app.AppCompatActivity;
import androidx.core.app.ActivityCompat;
import java.io.IOException;
import java.io.OutputStream;
import java.util.UUID;

public class MainActivity extends AppCompatActivity {

//Criação dos meus objetos
    Switch gps, leds, sound, smoke;
    Button sendConfig, setCoordinates;
    ImageButton playPause;
    ImageView teamLogo;
    MediaPlayer mediaPlayer = new MediaPlayer();
    EditText timeBomb, timeGame, numPlayers, bombCode;
    boolean musicSound;

    private BluetoothAdapter bluetoothAdapter;
    private BluetoothSocket bluetoothSocket;
    private BluetoothDevice bluetoothDevice;
```

```
    private OutputStream outputStream;

    private final String DEVICE_ADDRESS = "08:A6:F7:20:B3:1E"; //
Coloque o endereço MAC do seu módulo Bluetooth
    private final UUID MY_UUID = UUID.fromString("00001101-0000-1000-
8000-00805F9B34FB");

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

//Declaração dos meus objetos
        bluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
        bluetoothDevice =
bluetoothAdapter.getRemoteDevice(DEVICE_ADDRESS);

        mediaPlayer = MediaPlayer.create(this, R.raw.intro);
        mediaPlayer.start();
        playPause = (ImageButton) findViewById(R.id.play_pause_intro);
        gps = (Switch) findViewById(R.id.sw_gps);
        leds = (Switch) findViewById(R.id.sw_leds);
        smoke = (Switch) findViewById(R.id.sw_smoke);
        sound = (Switch) findViewById(R.id.sw_sound);
        sendConfig = (Button) findViewById(R.id.btn_send_config);
        setCoordinates = (Button) findViewById(R.id.setCoordinates);
        timeGame = (EditText) findViewById(R.id.edit_text_time_game);
        timeBomb = (EditText) findViewById(R.id.edit_text_time_bomb);
        numPlayers = (EditText) findViewById(R.id.edit_text_players);
        bombCode = (EditText) findViewById(R.id.edit_text_bomb_Code);


//Envia o tempo de jogo
        timeGame.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                String message = "TIMEG" +
timeGame.getText().toString() + "\n";
                sendData(message);
            }
        });

//Envia o confirmação para configuração de coordenadas
        setCoordinates.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                String message = "setCoordinates" + "\n";
                sendData(message);
            }
        });

//Envia o tempo da bomba
        timeBomb.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                String message = "TIMEB" +
timeGame.getText().toString() + "\n";
                sendData(message);
            }
        });
```

```
//Envia o número de jogadores
        numPlayers.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                String message = "TEAM" +
numPlayers.getText().toString() + "\n";
                sendData(message);
            }
        });


//Envia o código da bomba
        bombCode.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                if (bombCode.length() ==
Integer.parseInt(numPlayers.getText().toString())){
                    String message = "CODE" +
bombCode.getText().toString() + "\n";
                    sendData(message);
                }else{
                    Toast.makeText(MainActivity.this,
                            "Bomb Code Size let be the same of Players
Number",
                            Toast.LENGTH_LONG).show();
                }
            }
        });


        playPause.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                introSound();
            }
        });


//Controla a activação ou desativação do gps
        gps.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                if (gps.isChecked()) {
                    String message = "gpsON\n";
                    sendData(message);
                } else {
                    String message = "gpsOFF\n";
                    sendData(message);
                }
            }
        });
//Controla a activação ou desativação dos leds
        leds.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                String message;
                if (leds.isChecked()) {
                    leds.setText(R.string.switch_leds_on);
                    message = "ledsON\n";
```

```
                } else {
                    leds.setText(R.string.switch_leds_off);
                    message = "ledsOFF\n";

                }
                sendData(message);
            }
        });
//Controla a activação ou desativação do fumo
        smoke.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                String message;
                if (smoke.isChecked()) {
                    smoke.setText(R.string.switch_smoke_on);
                    message = "smokeON\n";
                } else {
                    smoke.setText(R.string.switch_smoke_off);
                    message = "smokeOFF\n";
                }
                sendData(message);
            }
        });
//Controla a activação ou desativação do som da bomba
        sound.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                String message;
                if (sound.isChecked()) {
                    sound.setText(R.string.switch_sound_on);
                    message = "soundON\n";
                } else {
                    sound.setText(R.string.switch_sound_off);
                    message = "soundOFF\n";
                }
                sendData(message);
            }
        });
//Envia a confirmação da configuração
        sendConfig.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                String message = "sendConfig";
                sendData(message);
            }
        });

        connectBluetooth();

    }
//Controlo do audio da aplicação
    public void introSound(){
        if(musicSound){
            musicSound = false;
            playPause.setImageResource(R.drawable.ic_play);
        }else{
            musicSound = true;
            playPause.setImageResource(R.drawable.ic_pause);
        }
        if(musicSound) {
```

```
            mediaPlayer.start();
        }else{
            mediaPlayer.pause();
        }
    }
//conecta ao bluetooth do macAddress indicado
    private void connectBluetooth() {
        try {
            bluetoothSocket =
bluetoothDevice.createRfcommSocketToServiceRecord(MY_UUID);
            bluetoothSocket.connect();
            outputStream = bluetoothSocket.getOutputStream();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
//Envia a menssagem
    private void sendData(String message) {
        try {
            outputStream.write(message.getBytes());
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
        try {
            bluetoothSocket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Código Arduíno

Devido ao projeto ser complexo e ter demasiadas linhas de código o mesmo foi dividido em vários ficheiros, criou-se várias funções para evitar repetir código e criou-se um objeto para armazenar as principais características do projeto.

Montain_Wolves_Bomb

```
#include <Wire.h>
#include "BluetoothSerial.h"
#include <LCD-I2C.h>
#include <Keypad.h>
#include "Bomb.h"
#include <Adafruit_SSD1306.h>
#include <TinyGPS++.h>
#include <FastLED.h>
#include "paletas.h"
#include <LoRa.h>

// Check if Bluetooth is available
#if !defined(CONFIG_BT_ENABLED) || !defined(CONFIG_BLUEDROID_ENABLED)
#error Bluetooth is not enabled! Please run `make menuconfig` to and
enable it
#endif
// Check Serial Port Profile
#if !defined(CONFIG_BT_SPP_ENABLED)
#error Serial Port Profile for Bluetooth is not available or not enabled.
It is only available for the ESP32 chip.
#endif

#define SCREEN_WIDTH 128 // OLED display width
#define SCREEN_HEIGHT 64 // OLED display height
#define OLED_RESET -1 //Reset pin
#define SCREEN_ADDRESS 0x3C //LCD Oled Address
#define NUM_LEDS_FITA 20
#define PINO_FITA 15
#define LED_BRIGHTNESS 10
#define RXD2 16
#define TXD2 17
#define beep 2

//external functions
extern void setup_ori();
extern void core_1();

HardwareSerial neogps(1);//GPS created serial


//Led Strip values
CRGB fita[NUM_LEDS_FITA];
uint8_t hue = 10;
uint8_t palleteIndex = 0;
uint16_t breatheLevel = 0;
byte cor = 0;
```

```
uint16_t beatA = 0;
uint16_t beatB = 0;
uint8_t sinBeat = 0;
uint8_t sinBeat2 = 0;
uint8_t sinBeat3 = 0;


unsigned long start = millis();
boolean setupFinish = false;

//Bluetooth device name
String device_name = "Montain_Wolves_Airsoft_Bomb";
boolean Configured = false;
boolean bluetoothConfigured = false;
const byte ROWS = 4; //four rows
const byte COLS = 4; //three columns
byte rowPins[ROWS] = {13, 12, 14,27}; //connect to the row pinouts of the keypad
byte colPins[COLS] = {26, 25, 33, 32}; //connect to the column pinouts of the keypad

char keys[ROWS][COLS] = {
  {'1','2','3','A'},
  {'4','5','6','B'},
  {'7','8','9','C'},
  {'*','0','#','D'}
};

TaskHandle_t Task2; //New task for initialized in core 0

BluetoothSerial BT; //Bluetooth object
Keypad keypad = Keypad(makeKeymap(keys), rowPins, colPins, ROWS, COLS ); //keypad object
LCD_I2C lcd(0x27, 16, 2); //liquid cristal lcd object
Bomb bomb(15, "123456789", 3, 3, 30); //bomb object
TinyGPSPlus gps; //gps object
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);
//oled display object




void setup() {
  setup_ori();

}

void loop() {
```

```
 core_1();
}
```

Setup

```
//External functions
extern void beepingTimes(int, int);

void setup_ori() {
  bomb.bombStatus = initialize;
  pinMode(beep, OUTPUT);
  beepingTimes(3, 250);
  Serial.begin(115200);

// SSD1306_SWITCHCAPVCC = generate display voltage from 3.3V internally
    if(!display.begin(SSD1306_SWITCHCAPVCC, SCREEN_ADDRESS)) {
      Serial.println(F("SSD1306 allocation failed"));
      for(;;); // Don't proceed, loop forever, ESP will crash and reboot
      }

  Wire.begin();
  lcd.begin(&Wire);
  lcd.display();
  lcd.backlight();
  display.clearDisplay();
  display.display();

  neogps.begin(9600, SERIAL_8N1, RXD2, TXD2);

  BT.begin(device_name);

  FastLED.addLeds<WS2812B, PINO_FITA, GRB>(fita, NUM_LEDS_FITA);
  FastLED.setBrightness(LED_BRIGHTNESS);

  delay(2000);

  display.display();
  Serial.printf("The device with name \"%s\" is started \nNow you can pai
it with Bluetooth\n", device_name.c_str());
  Serial.print("MAC Address: "); Serial.println(BT.getBtAddressString());

//Task core 0
  xTaskCreatePinnedToCore(
    gpsTracker,
    "Task2",
    10000,
    NULL,
```

```
    1,
    &Task2,
    0);

};
```

Objeto Bomb

Class

```
#pragma once

enum BombStatus {
  initialize,
  configuration,
  readyToArm,
  armed,
  disarm,
  explode
};


class Bomb{
  public:
    int time;
    int tries;
    int tryArming;
    int players;
    int gameTime;
    int speedLight = 20;
    const int speedSound[4] = {250, 500, 1000, 2000};
    BombStatus bombStatus = initialize;
    double latZone;
    double longZone;
    double maxDistance;
    boolean codeDiscovered;
    String code;
    boolean gps;
    boolean sound;
    boolean leds;
    boolean smoke;
    String checkGPS();
    String checkLeds();
    String checkSound();
    String checkSmoke();
    Bomb(int, String, int, int, int);
    boolean checkCode(String code);
```

```
    boolean isExplode();
    boolean finishGame(unsigned long);
    boolean isTimeOut(unsigned long);
    int getSize();
    boolean isValidZone();

  private:

};
```

Implementação

```
#include"Bomb.h"
extern double getDistance(double, double);


Bomb::Bomb(int time, String code, int tries, int tryArming, int
gameTime){
  this->time = time;
  this->tries = tries;
  this->tryArming = tryArming;
  this->code = code;
  this->players = 0;
  this->gameTime = gameTime * 60000;
  this->gps = false;
  this->sound = true;
  this->leds = true;
  this->smoke = false;
  this->latZone = 0.0;
  this->longZone = 0.0;
  this->maxDistance = 10.0;
  this->codeDiscovered = false;
};


String Bomb::checkGPS(){
  if (this->gps){
    return "Enable";
  }else {return "Disable";}
}

String Bomb::checkLeds() {
  if (this->leds){
    return "Enable";
  }else {return "Disable";}
}
```

```
String Bomb::checkSound() {
  if (this->sound){
    return "Enable";
  }else {return "Disable";}
}

String Bomb::checkSmoke() {
  if (this->smoke){
    return "Enable";
  }else {return "Disable";}
}

boolean Bomb::checkCode(String trycode){
  if (trycode == this->code){
    return true;
  }else{
    return false;
  }
};

boolean Bomb::isExplode(){
  if (this->tries < 0){
    return true;
  }else{
    return false;
  }
};

boolean Bomb::finishGame(unsigned long timeInit) {
  if (millis() - timeInit >= this->gameTime){
    return true;
  }
  return false;
}


boolean Bomb::isTimeOut(unsigned long timeInit) {
  if (millis() - timeInit >= this->time){
    return true;
  }
  return false;
}

int Bomb::getSize(){
  int size = this->code.length();
  return size;
};
```

```
boolean Bomb::isValidZone(){
  double distance = getDistance(this->latZone, this->longZone);
  if(distance <= bomb.maxDistance){
    return true;
  }else{
    return false;
  }
};
```

Bluetooth

```
extern boolean bluetoothConfigured;
extern void bombIsReady();
extern void beepingTimes(int, int);
extern void WaitSat();
extern void setNewCoordinates();
void bluetoothConfig(BluetoothSerial);
void bluetoothInit();

void bluetoothInit(){
  bomb.gps = false;
  bomb.sound = false;
  bomb.leds = false;
  bomb.smoke = false;
}

void bluetoothConfig(BluetoothSerial BT){
  String message = "";
  while(!bluetoothConfigured){
    (gps.satellites.value() < 3) ? fillSolidColor(CRGB::Red) :
fillSolidColor(CRGB::Green);

    if (BT.available()) {
      char inComingChar = BT.read();
    if (inComingChar != '\n'){
      message += String (inComingChar);
    }
    else{
      message = "";
    }
    Serial.write(inComingChar);

  if (message.substring(0, 4) == "TEAM")
  {
    if(bomb.sound){beepingTimes(5, 50);}
    bomb.players = message.substring(4).toInt();
    Serial.print("\nPlayers: "); Serial.println(bomb.players);
```

```
  lcd.clear();
  lcd.setCursor(4,0);
  lcd.print("Players");
  lcd.setCursor(7, 1);
  lcd.print(String(bomb.players));
}

if (message.substring(0,5) == "TIMEG")
{
  if(bomb.sound){beepingTimes(5, 50);}
  bomb.gameTime = message.substring(5).toInt() * 60000;
  Serial.print("\nGame Time: "); Serial.println(bomb.gameTime);
  lcd.clear();
  lcd.setCursor(3,0);
  lcd.print("Game Time");
  lcd.setCursor(2, 1);
  lcd.print(String(bomb.gameTime / 60000));
  lcd.setCursor(5, 1);
  lcd.print("Minutes");
}

  if (message.substring(0,5) == "TIMEB")
{
  if(bomb.sound){beepingTimes(5, 50);}
  bomb.time = message.substring(5).toInt() * 60000;
  Serial.print("\nGame Time: "); Serial.println(bomb.time);
  lcd.clear();
  lcd.setCursor(3,0);
  lcd.print("Bomb Time");
  lcd.setCursor(2, 1);
  lcd.print(String(bomb.time / 60000));
  lcd.setCursor(5, 1);
  lcd.print("Minutes");
}

if (message.substring(0,4) == "CODE")
{
  if(bomb.sound){beepingTimes(5, 50);}
  bomb.code = message.substring(4);
  Serial.print("Bomb Code: "); Serial.println(bomb.code);
  lcd.clear();
  lcd.setCursor(3,0);
  lcd.print("Bomb Code");
  lcd.setCursor(4, 1);
  lcd.print("Defined");
}
```

```
if (message == "gpsON")
{
  if(bomb.sound){beepingTimes(5, 50);}
  bomb.gps = true;
  Serial.println("\nGPS is ON!");
  lcd.clear();
  lcd.setCursor(6,0);
  lcd.print("GPS");
  lcd.setCursor(7, 1);
  lcd.print("ON");
}

if (message == "gpsOFF")
{
  if(bomb.sound){beepingTimes(5, 50);}
  bomb.gps = false;
  Serial.println("\nGPS is OFF!");
  lcd.clear();
  lcd.setCursor(6,0);
  lcd.print("GPS");
  lcd.setCursor(6, 1);
  lcd.print("OFF");
}

if (message == "setCoordinates") {
  if(bomb.sound){beepingTimes(5, 50);}
  if(gps.satellites.value() > 5){
    bomb.latZone = gps.location.lat();
    bomb.longZone = gps.location.lng();
    setNewCoordinates();
  } else {
    WaitSat();
    beepingTimes(20, 50);
    delay(2000);
    lcd.clear();
  }
}

if (message == "ledsON")
{
  if(bomb.sound){beepingTimes(5, 50);}
  bomb.leds = true;
  Serial.println("\nLeds are ON!");
  lcd.clear();
  lcd.setCursor(6,0);
  lcd.print("LEDS");
  lcd.setCursor(7, 1);
```

```
    lcd.print("ON");
  }

  if (message == "ledsOFF")
  {
    if(bomb.sound){beepingTimes(5, 50);}
    bomb.leds = false;
    Serial.println("\nLeds are OFF!");
    lcd.clear();
    lcd.setCursor(6,0);
    lcd.print("LEDS");
    lcd.setCursor(6, 1);
    lcd.print("OFF");
  }

  if (message == "soundON")
  {
    if(bomb.sound){beepingTimes(5, 50);}
    bomb.sound = true;
    Serial.println("\nSound is ON!");
    lcd.clear();
    lcd.setCursor(5,0);
    lcd.print("SOUND");
    lcd.setCursor(7, 1);
    lcd.print("ON");
  }

  if (message == "soundOFF")
  {
    if(bomb.sound){beepingTimes(5, 50);}
    bomb.sound = false;
    Serial.println("\nSound is OFF!");
    lcd.clear();
    lcd.setCursor(6,0);
    lcd.print("SOUND");
    lcd.setCursor(6, 1);
    lcd.print("OFF");
  }

  if (message == "smokeON")
  {
    if(bomb.sound){beepingTimes(5, 50);}
    bomb.smoke = false;
    Serial.println("\nSmoke is ON!");
    lcd.clear();
    lcd.setCursor(5,0);
    lcd.print("SMOKE");
```

```
    lcd.setCursor(7, 1);
    lcd.print("ON");
  }

  if (message == "smokeOFF")
  {
    if(bomb.sound){beepingTimes(5, 50);}
    bomb.smoke = false;
    Serial.println("\nSmoke is OFF!");
    lcd.clear();
    lcd.setCursor(5,0);
    lcd.print("SMOKE");
    lcd.setCursor(6, 1);
    lcd.print("OFF");
  }

  if (message == "sendConfig")
  {
    if(bomb.sound){beepingTimes(1, 2000);}
    if(bomb.gps && bomb.latZone != 0.0 || !bomb.gps) {
      bluetoothConfigured = true;
      Serial.println("\nGame READY!!");
      lcd.clear();
      bombIsReady();
      delay(2000);
      lcd.clear();
    } else {
      lcd.setCursor(6, 0);
      lcd.print("Set");
      lcd.setCursor(2, 1);
      lcd.print("Coordinates");
      delay(2000);
    }

  }
  delay(50);
  }
 }
}
```

ManuallyConfig

```
boolean manuallyConfigured();
extern void checkStatus(boolean);
extern boolean isNum(char);
extern void printPlayers();
extern void printTimeGame();
```

```
extern void printTimeBomb();
extern void printBombCode();
extern void printDigit(String);
extern void printSmoke(String);
extern void printSound(String);
extern void printLeds(String);
extern void printGPS(String);
extern void printConfirmConfig();
extern void beepingTimes(int, int);

enum Config {
  Players,
  TimeGame,
  TimeBomb,
  BombCode,
  GPS,
  SetNewCoordinate,
  Leds,
  Sound,
  Smoke,
  Confirm
};

boolean manuallyConfigured(){
  Config config = Players;
  String _players = "";
  String _timeGame = "30";
  String _timeBomb = "15";
  String _code = "";

  char key;
  while (!Configured) {
    switch(config){
      case Players: {
        printPlayers();
        printDigit(_players);
        key = keypad.getKey();
        if (key) {
          beepingTimes(1, 50);
          if (isNum(key)) {
            //lcd.clear();
            _players += key;
            printDigit(_players);
            }
          if (key == 'C' && _players.length() > 0) {
            lcd.clear();
            _players = _players.substring(0, _players.length() - 1);
```

```
            printDigit(_players);
            }
          if (key == 'D' && _players.length() > 0) {
            lcd.clear();
            config = TimeGame;
            bomb.players = _players.toInt();
            }
          }
        break;
        }
        case TimeGame: {
          printTimeGame();
          printDigit(_timeGame);
          key = keypad.getKey();
          if (key) {
            beepingTimes(1, 50);
            if (isNum(key)) {
              //lcd.clear();
              _timeGame += key;
              printDigit(_timeGame);
            }
            if (key == 'C' && _timeGame.length() > 0) {
              lcd.clear();
              _timeGame = _timeGame.substring(0, _timeGame.length() - 1);
              printDigit(_timeGame);
            }
            if (key == 'D' && _timeGame.length() > 0) {
              bomb.gameTime = _timeGame.toInt();
              bomb.gameTime *= 60000;
              config = TimeBomb;
              lcd.clear();
            }
          }
          break;
        }
        case TimeBomb : {
        printTimeBomb();
        printDigit(_timeBomb);
        key = keypad.getKey();
        if (key) {
          beepingTimes(1, 50);
          if (isNum(key)) {
            //lcd.clear();
            _timeBomb += key;
            printDigit(_timeBomb);
          }
          if (key == 'C' && _timeBomb.length() > 0) {
```

```
            lcd.clear();
            _timeBomb = _timeBomb.substring(0, _timeBomb.length() - 1);
            printDigit(_timeBomb);
          }
          if (key == 'D' && _timeBomb.length() > 0) {
            bomb.time = _timeBomb.toInt() * 60000;
            config = BombCode;
            lcd.clear();
          }
        }
        break;
        }

        case BombCode: {
        printBombCode();
        printDigit(_code);
        key = keypad.getKey();
        if (key) {
          beepingTimes(1, 50);
          if (isNum(key)) {
            //lcd.clear();
            _code += key;
            printDigit(_code);
            }
          if (key == 'C' && _code.length() > 0) {
            lcd.clear();
            _code = _code.substring(0, _code.length() - 1);
            printDigit(_code);
            }
          if (key == 'D' && _code.length() > 0 && _code.length() ==
_players.toInt()) {
            bomb.code = _code;
            config = GPS;
            lcd.clear();
            }
          }
          break;
        }

        case GPS: {
          checkStatus(bomb.gps);
          printGPS(bomb.checkGPS());
          key = keypad.getKey();
          if (key) {
            beepingTimes(1, 50);
            if (key == 'B' && bomb.gps) {
              lcd.clear();
```

```
            bomb.gps = false;
          }
          if (key == 'A' && !bomb.gps) {
            lcd.clear();
            bomb.gps = true;
          }
          if (key == 'D') {
            bomb.gps ? config = SetNewCoordinate : config = Leds;
            lcd.clear();
          }
        }
        break;
      }

      case SetNewCoordinate: {
        checkStatus(gps.satellites.value() > 6);
        gps.satellites.value() < 6 ? WaitSat() : printCoordinate();
        key = keypad.getKey();
        if (key) {
          beepingTimes(1, 50);
          if (key == 'A') {
            lcd.clear();
            setNewCoordinates();
            bomb.latZone = gps.location.lat();
            bomb.longZone = gps.location.lng();
            delay(2000);
            }
          if (key == 'D') {
            config = Leds;
            lcd.clear();
          }
        }
        break;
      }

      case Leds: {
        checkStatus(bomb.leds);
        printLeds(bomb.checkLeds());
        key = keypad.getKey();
        if (key) {
          beepingTimes(1, 50);
          if (key == 'B' && bomb.leds) {
            lcd.clear();
            bomb.leds = false;
          }
          if (key == 'A' && !bomb.leds) {
            lcd.clear();
```

```
            bomb.leds = true;
          }
          if (key == 'D') {
            config = Sound;
            lcd.clear();
          }
        }
        break;
      }

      case Sound: {
        checkStatus(bomb.sound);
        printSound(bomb.checkSound());
        key = keypad.getKey();
        if (key) {
          beepingTimes(1, 50);
          if (key == 'B' && bomb.sound) {
            lcd.clear();
            bomb.sound = false;
          }
          if (key == 'A' && !bomb.sound) {
            lcd.clear();
            bomb.sound = true;
          }
          if (key == 'D') {
            config = Confirm;
            //config = Smoke;
            lcd.clear();
          }
        }
        break;
      }

      case Smoke: {
        checkStatus(bomb.smoke);
        printSmoke(bomb.checkSmoke());
        key = keypad.getKey();
        if (key) {
          beepingTimes(1, 50);
          if (key == 'B' && bomb.smoke) {
            lcd.clear();
            bomb.smoke = false;
          }
          if (key == 'A' && !bomb.smoke) {
            lcd.clear();
            bomb.smoke = true;
          }
```

```
            if (key == 'D') {
              config = Confirm;
              lcd.clear();
            }
          }
          break;
        }

        case Confirm: {
          printConfirmConfig();
          key = keypad.getKey();
          if (key) {
            beepingTimes(1, 50);
            if (key == 'C') {
              lcd.clear();
              config = Players;
            }
            if (key == 'D') {
              lcd.clear();
              Configured = true;
              fillSolidColor(CRGB::Black);
            }
          }
          break;
        }
      }
    }
    return true;
  }
```

Core0

```
extern void respiracao(byte, accum88);
extern void gradienteOndas();
extern void fillSolidColor(CRGB);
extern void gradienteMovendo();
extern void luzesDancantes();
extern void cometa();
extern void getData();
```

```
extern void SerialGPSData();
extern boolean getNewData();
extern void clearAll();
extern void NoData();
extern void printDisplayData();

void gpsTracker(void * pvParameters) {
  for(;;) {
    if(gps.satellites.value() < 1) {
      display.clearDisplay();
    }
    if(getNewData()) {
      if(gps.satellites.value() > 3) {
        SerialGPSData();
        printDisplayData();
      }
    }
    vTaskDelay(pdMS_TO_TICKS(10));
  }
}
```

Core1

```
extern void teamIntro();
extern void bluetoothInit();
extern void gradienteOndas();
extern void printBluetoothChoice();
extern void printBluetoothConfig();
extern void bluetoothConfig(BluetoothSerial);
extern boolean manuallyConfigured();
extern void invalidZone();
extern void availableZone();
extern void insertCode();
extern void correctCode();
extern void wrongCode();
extern void tryAgain();
extern void bombClock(unsigned long, int, int);
extern void youWin();
extern void bombArmed();
extern void printDigit(String);
extern void printCode(String);
extern void bombExploded();
```

```
extern void bombExplodedToArming();
extern void restart();
extern void beepingTimes(int, int);
extern void beepBomb();
extern void beepOn(boolean);
void core_1();
unsigned long lastBeep;
int bombTime;

String code = "";
String secondCode = "";
int codeSize = 0;
int secondCodeSize = 0;
int minutesAnterior = bomb.time / 1000;
int secondsAnterior = 59;

unsigned long int bombTime_millisAnterior = 0;
unsigned long int clock_millisAnterior = 0;
unsigned long int gameTimeLast = 0;
int count = 0;
int timeMin;
int timeSec;

enum GameStatus {
  Intro,
  Configuration,
  Prepared,
  ReadyToArm,
  TryCode,
  VerifyCode,
  Disarm,
  Explode,
  ExplodeTryArming,
  Restart
};

GameStatus gameStatus = Intro;

void core_1(){
  switch(gameStatus){
    case Intro: {
      fillSolidColor(CRGB::Blue);
      char key = keypad.getKey();
      key ? beepingTimes(1, 50): void();
      teamIntro();
      if (key == 'D') {
        lcd.clear();
```

```
      gameStatus = Configuration;
      fillSolidColor(CRGB::Black);
    }
    break;
  }

  case Configuration: {
    bomb.bombStatus = configuration;
    printBluetoothChoice();
    char key = keypad.getKey();
    if (key == 'A'){
      beepingTimes(1, 50);
      lcd.clear();
      String message = "";
      bluetoothInit();
      printBluetoothConfig();
      bluetoothConfig(BT);
      lcd.clear();
      gameStatus = Prepared;
      gameTimeLast = millis();
    }
    if (key == 'D'){
      beepingTimes(1, 50);
      lcd.clear();
      if (manuallyConfigured()) {
        gameStatus = Prepared;
      }
    }
    break;
  }

  case Prepared: {
    if(bomb.gps) {
      if(bomb.isValidZone()){
        availableZone();
        char key = keypad.getKey();
        if(key == 'D') {
          gameStatus = ReadyToArm;
          lcd.clear();
        }
      }else{
        lcd.setCursor(0,0);
        invalidZone();
      }
    } else {
      gameStatus = ReadyToArm;
    }
```

```
      break;
    }

    case ReadyToArm: {
      bomb.bombStatus = readyToArm;
      if(!bomb.isValidZone() && bomb.gps) {
        gameStatus = Prepared;
      }
      if (millis() - gameTimeLast >= bomb.gameTime) {
        bomb.bombStatus = explode;
        gameStatus = Explode;
        lcd.clear();
        break;
      }
      insertCode();
      char key = keypad.getKey();
      if(key){
        beepingTimes(1, 50);
        if(key == 'D' && bomb.tryArming > 0 && codeSize > 0){
          if(code == bomb.code){
            gameStatus = TryCode;
            Serial.println("https://www.google.com/maps/place/" +
String(bomb.latZone,6) + "," + String(bomb.longZone,6));
            lastBeep = millis();
            timeMin = bomb.time / 60000;
            bombTime = bomb.time;
            timeSec = 0;
            lcd.clear();
            bomb.bombStatus = armed;
          }else{
            lcd.clear();
            wrongCode();
            tryAgain();
            delay(500);
            bomb.tryArming--;
            code = "";
            codeSize = 0;
            delay(500);
            lcd.clear();
            if (bomb.tryArming == 0){
              lcd.clear();
              gameStatus = ExplodeTryArming;
            }
          }
        } else if (key == 'C' && codeSize > 0){
          lcd.clear();
          codeSize--;
```

```
          if (codeSize > 0) {
            code = code.substring(0, code.length() - 1);
            printCode(code);
          }else {
            code = "";
          }
        }else  if (isNum(key)) {
          code += key;
          codeSize++;
          printDigit(code);
        }
      }
      break;
    }

    case TryCode: {
      bomb.leds ? respiracao(22, bomb.speedLight) :
fillSolidColor(CRGB::Black);
      if (millis() - gameTimeLast >= bomb.gameTime) {
        bomb.bombStatus = explode;
        gameStatus = Explode;
        lcd.clear();
        break;
      }
      beepBomb();
      printDigit(secondCode);
      bombArmed();
      if (bomb.time > 0) {
        if (timeSec == 0) {
          bombTime_millisAnterior = millis();
          timeMin--;
          timeSec = 59;
          //lcd.clear();
        } else if (millis() - bombTime_millisAnterior >= 1000) {
          bombTime_millisAnterior = millis();
          timeSec--;
          bomb.time -= 1000;
          //lcd.clear();
        }
        if (timeMin > 0 || timeSec > 0){
          if(timeSec == 9 || timeMin == 9) {lcd.clear();}
          printClock(timeMin, timeSec);
        }
      }else {
        lcd.clear();
        gameStatus = Explode;
      }
```

```
      char key = keypad.getKey();
      if (key){
        beepingTimes(1, 50);
        if (key == 'D' && secondCodeSize > 0){
          gameStatus = VerifyCode;
          lcd.clear();
        }else if (key == 'C' && secondCodeSize > 0){
          lcd.clear();
          secondCodeSize--;
          if (secondCodeSize > 0) {
            secondCode = secondCode.substring(0, secondCode.length() -
1);
            printCode(secondCode);
          }else{
            secondCode = "";
          }
        }else if (isNum(key)){
          secondCode += key;
          secondCodeSize++;
          printDigit(secondCode);
        }
      }
      break;
    }

    case VerifyCode: {
      if(bomb.checkCode(secondCode)){
        lcd.clear();
        correctCode();
        bomb.codeDiscovered = true;
        gameStatus = Disarm;
        lcd.clear();
      } else if (bomb.tries == 0){
        lcd.clear();
        bomb.codeDiscovered = false;
        gameStatus = Explode;
      }else{
        lcd.clear();
        bomb.codeDiscovered = false;
        wrongCode();
        tryAgain();
        bomb.tries--;
        bomb.leds ? fillSolidColor(CRGB::Red) :
fillSolidColor(CRGB::Black);
        (bomb.tries > 0) ? delay(5 * 1000) : delay(0);
        fillSolidColor(CRGB::Black);
```

```
        secondCodeSize = 0;
        secondCode = "";
        lcd.clear();
        gameStatus = TryCode;
      }
      break;
    }

    case Disarm: {
      bomb.leds ? fillSolidColor(CRGB::Green) :
fillSolidColor(CRGB::Black);
      bomb.bombStatus = disarm;
      youWin();
      beepOn(bomb.sound);
      char key = keypad.getKey();
      if (key == 'D') {
        gameStatus = Restart;
        lcd.clear();
        fillSolidColor(CRGB::Black);
        delay(200);
      }
      break;
    }

    case Explode: {
      bomb.leds ? fillSolidColor(CRGB::Red) :
fillSolidColor(CRGB::Black);
      bomb.bombStatus = explode;
      beepOn(bomb.sound);
      bombExploded();
      char key = keypad.getKey();
      if (key == 'D') {
        gameStatus = Restart;
        lcd.clear();
        fillSolidColor(CRGB::Black);
        delay(200);
      }
      break;
    }

    case ExplodeTryArming: {
      bomb.leds ? fillSolidColor(CRGB::Red) :
fillSolidColor(CRGB::Black);
      beepOn(bomb.sound);
      bombExplodedToArming();
      char key = keypad.getKey();
      if (key == 'D') {
```

```
        lcd.clear();
        gameStatus = Restart;
        fillSolidColor(CRGB::Black);
        delay(200);
      }
      break;
    }

    case Restart: {
      char key = keypad.getKey();
      restart();
      digitalWrite(beep, LOW);
      if (key) {
        beepingTimes(1, 50);
        if (key == 'D') {
          fillSolidColor(CRGB::Black);
          beepOn(false);
          ESP.restart();
        }
      }
      break;
    }

    default: {
      Serial.println("Something wrong!!");
      break;
    }
  }
}
```

Beep

```
void beepingTimes(int, int);
void beepBomb();
void beepOn(boolean);
extern unsigned long lastBeep;
extern int bombTime;
```

```
void beepingTimes(int times, int time){
  if (bomb.sound) {
    for (int i = 0; i < times; i++) {
      digitalWrite(beep, HIGH);
      delay(time);
      digitalWrite(beep, LOW);
      delay(time);
    }
  }
}

void beepBomb() {
  int bombState = 0;
  if (bomb.time > bombTime / 2){
    //bombState = 2000;
    bombState = bomb.speedSound[3];
  }else if (bomb.time < bombTime / 2 && bomb.time > bombTime / 4) {
    bomb.speedLight = 50;
    //bombState = 1000;
    bombState = bomb.speedSound[2];
  } else if (bomb.time < bombTime / 4 && bomb.time > 20000) {
    bomb.speedLight = 100;
    //bombState = 500;
    bombState = bomb.speedSound[1];
  } else if (bomb.time < 20000) {
    bomb.speedLight = 255;
    //bombState = 250;
    bombState = bomb.speedSound[0];
  }
  if (millis() - lastBeep >= bombState && bomb.sound) {
    if (digitalRead(beep) == HIGH) {
      lastBeep = millis();
      digitalWrite(beep, LOW);
    }else {
      digitalWrite(beep, HIGH);
      lastBeep = millis();
    }
  } else if (millis() - lastBeep < bombState && millis() - lastBeep >= 50
&& bomb.sound) {
      int stateBeeper = digitalRead(beep);
      if (stateBeeper == HIGH) {
        digitalWrite(beep, LOW);
      }
  }
}
```

```
void beepOn(boolean beeping) {
  if (beeping) {
    delay(10);
    digitalWrite(beep, HIGH);
  } else {
    digitalWrite(beep, LOW);
  }
}
```

Leds_WS2812b

```
void respiracao(byte);
void gradienteOndas();
void gradienteMovendo();
void fillSolidColor(CRGB);
void luzesDancantes();
void cometa();
void explosao();
CRGB returnColor(int);
CRGBPalette16 returnPalette(int);
void checkStatus();



void checkStatus(boolean status) {
  status ? fillSolidColor(CRGB::Green) : fillSolidColor(CRGB::Red);
}


/*
 * EFEITO 1
 */
void respiracao(byte selcor, accum88 speed) {
    breatheLevel = beatsin16(speed, 0, 255);
    fill_solid(fita, NUM_LEDS_FITA, CHSV(selcor, 255, breatheLevel));
    FastLED.show();
}

/*
 * EFEITO 2
 */
void gradienteOndas() {
  CRGBPalette16 selPalette = returnPalette(random(1, NUM_PALETTES + 1));
  int indexPal = 2;
  beatA = beatsin16(30, 0, 255);
  beatB = beatsin16(20, 0, 255);
```

```
  fill_palette(fita, NUM_LEDS_FITA, (beatA + beatB) / 2, indexPal,
selPalette, 255, LINEARBLEND);
  FastLED.show();

}


/*
 * EFEITO 3
 */
void gradienteMovendo() {
  CRGBPalette16 selPalette = returnPalette(random(1, NUM_PALETTES + 1));
  for (int i = 0; i < 200; i++) {
    fill_palette(fita, NUM_LEDS_FITA, palleteIndex, 255 / NUM_LEDS_FITA,
selPalette, 255, LINEARBLEND);
    FastLED.show();
    palleteIndex++;
    delay(10);
  }
}


/*
 * EFEITO 6
 */
void fillSolidColor(CRGB selcor) {
  fill_solid(fita, NUM_LEDS_FITA, selcor);
  FastLED.show();
}

/*
 * EFEITO 7
 */
void luzesDancantes() {
  int c = random(1, 5);
  CRGB cor1 = returnColor(c);
  CRGB cor2 = returnColor(c + 1);
  CRGB cor3 = returnColor(c + 2);

  int c2 = random(1, 5);
  CRGB cor4 = returnColor(c2);
  CRGB cor5 = returnColor(c2 + 1);
  CRGB cor6 = returnColor(c2 + 2);

  sinBeat   = beatsin8(30, 0, NUM_LEDS_FITA / 2, 0, 0);
  sinBeat2  = beatsin8(30, 0, NUM_LEDS_FITA / 2, 0, NUM_LEDS_FITA);
  sinBeat3  = beatsin8(30, 0, NUM_LEDS_FITA / 2, 0, NUM_LEDS_FITA / 2);
```

```
  fita[sinBeat]   = cor1;
  fita[sinBeat2]  = cor2;
  fita[sinBeat3]  = cor3;

  fita[sinBeat + NUM_LEDS_FITA / 2]   = cor4;
  fita[sinBeat2 + NUM_LEDS_FITA / 2]  = cor5;
  fita[sinBeat3 + NUM_LEDS_FITA / 2]  = cor6;

  fadeToBlackBy(fita, NUM_LEDS_FITA, 10);
  FastLED.show();
}

/*
 * EFEITO 7
 */
void cometa() {
  byte fade = 128;
  int cometaSize = 5;
  byte hue = random(1, 255);
  int iDirection = 1;
  int iPos = 0;

  for (int i = 0; i < 1000; i++) {

    iPos += iDirection;
    if (iPos == (NUM_LEDS_FITA - cometaSize) || iPos == 0)
      iDirection *= -1;

    for (int i = 0; i < cometaSize; i++)
      fita[iPos + i].setHue(hue);

    // Randomly fade the LEDs
    for (int j = 0; j < NUM_LEDS_FITA; j++)
      if (random(10) > 5)
        fita[j] = fita[j].fadeToBlackBy(fade);

    delay(20);
    FastLED.show();
  }
}

/*
 * EFEITO 8
 */
void explosao() {
  FastLED.clear();
  byte fade = 128;
```

```
  int expSize = NUM_LEDS_FITA / 2;    // tamanho da explosao
  int numExplosoes = 4;          // quantas explosoes no efeito

  for (int x = 0; x < numExplosoes ; x++) {
    byte hue = random(1, 255);         // escolhe cor aleatoria
    //Serial.print("Cor: "); Serial.println(hue);

    for (int i = 0; i < expSize; i++) {
      fita[NUM_LEDS_FITA / 2 + i].setHue(hue);
      fita[NUM_LEDS_FITA / 2 - i].setHue(hue);
      if (i > expSize / 2) {
        i++;
        fita[NUM_LEDS_FITA / 2 + i].setHue(hue);
        fita[NUM_LEDS_FITA / 2 - i].setHue(hue);
      }
      FastLED.show();
    }

    delay(10);
    for (int i = 0; i < 50; i++) {
      for (int j = 0; j < NUM_LEDS_FITA; j++) {
        if (random(10) > 8)
          fita[j] = fita[j].fadeToBlackBy(fade);
      }
      delay(10);
      FastLED.show();
    }
  }
}


/*
   FUNÇÃO DE SELEÇÃO DE CORES
*/
CRGB returnColor(int num) {
  switch (num) {
    case 1:
      return CRGB::Red;
      break;
    case 2:
      return CRGB::Green;
      break;
    case 3:
      return CRGB::Blue;
      break;
    case 4:
      return CRGB::Pink;
```

```
      break;
    case 5:
      return CRGB::Yellow;
      break;
    case 6:
      return CRGB::Magenta;
      break;
    case 7:
      return CRGB::Cyan;
      break;
    case 8:
      return CRGB::Purple;
      break;
    case 9:
      return CRGB::Gray;
      break;
    default:
      return CRGB::Black;
      break;
  }
}

/*
   FUNÇÃO DE SELEÇÃO DE PALETAS
*/
CRGBPalette16 returnPalette(int selPalette) {
  switch (selPalette) {
    case 1:
      return brownGreenPalette;
      break;
    case 2:
      return heatPalette;
      break;
    case 3:
      return purplePalette;
      break;
    case 4:
      return greenbluePalette;
      break;
    case 5:
      return sunsetPalette;
      break;
    case 6:
      return fireandicePalette;
      break;
    case 7:
      return turqPalette;
```

```
      break;
    case 8:
      return autumnrosePalette;
      break;
    case 9:
      return bhw1_06Palette;
      break;
    case 10:
      return xmasPalette;
      break;
    case 11:
      return justduckyPalette;
      break;
    default:
      return heatPalette;
      break;
  }
}
```

Gps

```
int fuseTime = 1;
void getData();
double getDistance();
void SerialGPSData();
void clearAll();


double getDistance(double latZone, double longZone){
  double distance = TinyGPSPlus::distanceBetween(gps.location.lat(),
gps.location.lng(), latZone, longZone);
  return distance;
}


boolean getNewData() {
  boolean newData = false;
  if (start - millis() >= 1000) {
    //Serial.print("inicio: "); Serial.println(millis());
    while (neogps.available())
    {
      if (gps.encode(neogps.read())) {
        newData = true;
        break;
      }
    }
    //Serial.print("Fim: "); Serial.println(millis());
```

```
  }
  return newData;
}


void SerialGPSData() {
  Serial.println(gps.satellites.value());
  Serial.print("LAT=");  Serial.println(gps.location.lat(), 6);
  Serial.print("LONG="); Serial.println(gps.location.lng(), 6);
  Serial.print("ALT=");  Serial.println(gps.altitude.meters());
  Serial.print("Time= "); Serial.print(gps.time.hour()+1);
Serial.print(":"); Serial.print(gps.time.minute()); Serial.print(":");
Serial.println(gps.time.second());
  Serial.print("Date= "); Serial.println(gps.date.value());
  Serial.print("Hdop value: "); Serial.println(gps.hdop.value());
  display.clearDisplay();
  display.setTextColor(SSD1306_WHITE);
}

void printDisplayData() {
  display.clearDisplay();
  display.setTextSize(1);
  display.setCursor(2, 5);
  display.print("Lat: "); display.print(gps.location.lat(),6);
  display.setCursor(2, 15);
  display.print("Long: "); display.print(gps.location.lng(),6);
  display.setCursor(2, 25);
  display.print("SAT: "); display.print(gps.satellites.value());
  display.setCursor(2, 35);
  display.print("speed: "); display.print(gps.speed.kmph());
  display.setCursor(2, 45);
  display.print("hdop: "); display.print(gps.hdop.value());
  display.setCursor(2, 55);
  double distance = TinyGPSPlus::distanceBetween(gps.location.lat(),
gps.location.lng(), bomb.latZone, bomb.longZone);
  display.print("Dist: "); display.print(distance);
  display.display();
}

void clearAll() {
  display.clearDisplay();
  delay(1000);
  display.display();
}


void NoData() {
  display.setTextColor(SSD1306_WHITE);
```

```
  display.setTextSize(1);
  display.setCursor(2, 5);
  display.print("Lat: ");
  display.setCursor(2, 15);
  display.print("Long: ");
  display.setCursor(2, 25);
  display.print("SAT: ");
  display.setCursor(2, 35);
  display.print("speed: ");
  display.setCursor(2, 45);
  display.print("hdop: "); display.print(gps.hdop.value());
  display.setCursor(2, 55);
  display.print("Dist: ");
  display.display();
}
```

Keypad

```
boolean isNum(char);


boolean isNum(char _key){
  switch (_key) {
    case '0': {
      return true;
      break;
    }
    case '1':  {
      return true;
      break;
    }
    case '2': {
      return true;
      break;
    }
    case '3': {
      return true;
      break;
    }
    case '4': {
      return true;
      break;
    }
    case '5':  {
      return true;
      break;
    }
```

```
    case '6': {
      return true;
      break;
    }
    case '7': {
      return true;
      break;
    }
    case '8': {
      return true;
      break;
    }
    case '9': {
      return true;
      break;
    }
    default :{
      return false;
    }
  }
}
```

Display_liquidCrystal

```
extern boolean setupFinish;
extern String code;

void printBluetoothConfig();
void printPlayers();
void printTimeGame();
void printTimeBomb();
void printBombCode();
void printSmoke(String);
void printSound(String);
void printLeds(String);
void printGPS(String);
void WaitSat();
void printCoordinate();
void printConfirmConfig();
void bombIsReady();
void availableZone();
void invalidZone();
void insertCode();
void teamIntro();
void printDigit(String);
void printCode(String);
void correctCode();
```

```
void wrongCode();
void tryAgain();
void bombExploded();
void bombExplodedToArming();
void youWin();
void bombArmed();
void printClock(int, int);
void restart();
void printBarChar();


uint8_t barChar[8] = {
  0b11111,
  0b11111,
  0b11111,
  0b11111,
  0b11111,
  0b11111,
  0b11111,
  0b11111,
  };


void printBluetoothChoice() {
  lcd.setCursor(3,0);
  lcd.print("Bluetooth");
  lcd.setCursor(0,1);
  lcd.print("A - Yes   D - No");
  }

void printBluetoothConfig(){
  lcd.setCursor(3,0);
  lcd.print("Bluetooth");
  lcd.setCursor(1,1);
  lcd.print("Configuration!");
}

void printPlayers() {
  lcd.setCursor(0,0);
  lcd.print("Players Number");
}

void printTimeGame() {
  lcd.setCursor(0,0);
  lcd.print("Time Game");
}

void printTimeBomb() {
```

```
  lcd.setCursor(0,0);
  lcd.print("Time Bomb");
}

void printBombCode() {
  lcd.setCursor(0,0);
  lcd.print("Bomb Code");
}

void printGPS(String status){
  lcd.setCursor(0,0);
  lcd.print("GPS?");
  lcd.setCursor(9,0);
  lcd.print(status);
  lcd.setCursor(0,1);
  lcd.print("A - Yes   B - No");
}

void WaitSat() {
  lcd.setCursor(0,0);
  lcd.print("*****Wait*******");
  lcd.setCursor(0, 1);
  lcd.print("For more Sats!!*");
}

void printCoordinate() {
  lcd.setCursor(0, 0);
  lcd.print("Lat: "); lcd.print(gps.location.lat(), 6);
  lcd.setCursor(0, 1);
  lcd.print("Long: "); lcd.print(gps.location.lng(), 6);
}

void setNewCoordinates() {
  lcd.setCursor(2, 0);
  lcd.print("coordenates");
  lcd.setCursor(3, 1);
  lcd.print("Configured");
}

void printLeds(String status){
  lcd.setCursor(0,0);
  lcd.print("Leds?");
   lcd.setCursor(9,0);
  lcd.print(status);
  lcd.setCursor(0,1);
  lcd.print("A - Yes   B - No");
```

```
}

void printSound(String status){
  lcd.setCursor(0,0);
  lcd.print("Sound?");
   lcd.setCursor(9,0);
  lcd.print(status);
  lcd.setCursor(0,1);
  lcd.print("A - Yes   B - No");

}

void printSmoke(String status){
  lcd.setCursor(0,0);
  lcd.print("Smoke?");
   lcd.setCursor(9,0);
  lcd.print(status);
  lcd.setCursor(0,1);
  lcd.print("A - Yes   B - No");
}

void printConfirmConfig(){
  lcd.setCursor(4,0);
  lcd.print("Confirm?");
  lcd.setCursor(0,1);
  lcd.print("D - Yes   C - No");
}

void bombIsReady(){
  lcd.setCursor(6,0);
  lcd.print("Bomb");
  lcd.setCursor(5, 1);
  lcd.print("READY!");
}

void availableZone(){
  lcd.backlight();
  lcd.setCursor(6, 0);
  lcd.print("Zone");
  lcd.setCursor(3, 1);
  lcd.print("Autorized");
}

void invalidZone(){
  //lcd.backlightOff();
  lcd.setCursor(6, 0);
  lcd.print("Zone");
```

```
  lcd.setCursor(4, 1);
  lcd.print("Invalid");
}

void teamIntro(){
  lcd.setCursor(1,0);
  lcd.print("Montain Wolves");
  lcd.setCursor(4,1);
  lcd.print(F("Press D!!"));
}


void insertCode(){
  lcd.setCursor(0,0);
  lcd.print("Insert Bomb Code");
}


void bombArmed(){
  lcd.setCursor(0,0);
  lcd.print("Armed");
}

void printClock(int timeMin, int timeSec){
  lcd.setCursor(11, 0); // Or setting the cursor in the desired position.
  if (timeMin < 10){
    lcd.print("0");
  }
  lcd.print(timeMin);
  lcd.print(":");
  if (timeSec < 10){
    lcd.print("0");
  }
  lcd.print(timeSec);
}

void printDigit(String digit){
  lcd.setCursor(0, 1);
  lcd.print(digit);
}

void correctCode(){
  lcd.setCursor(1,0);
  lcd.print("Correct Code!");
}

void wrongCode(){
  lcd.setCursor(2,0);
```

```
  lcd.print("Wrong Code!");
}

void printCode(String code){
  lcd.setCursor(0, 1);
  lcd.print(code);
}

void tryAgain(){
  lcd.setCursor(3,1);
  lcd.print("Try Again!!");
}

void bombExploded(){
  lcd.setCursor(1,0);
  lcd.print("Bomb Exploded");
}

void bombExplodedToArming(){
  lcd.setCursor(1,0);
  lcd.print("Bomb Exploded");
  lcd.setCursor(3, 1);
  lcd.print("To Arming");
}

void youWin(){
  lcd.setCursor(4,0);
  lcd.print("YOU WIN!");
}

void restart() {
  lcd.setCursor(5,0);
  lcd.print("Restart");
  lcd.setCursor(4,1);
  lcd.print("Press D");
}

void printBarChar(){
  lcd.createChar(0, barChar);
  lcd.write(0);
}
```

Paletas

```
#define NUM_PALETTES 11

DEFINE_GRADIENT_PALETTE( browngreen_gp ) {
```

```
  0,    6,  255,    0,     //green
  71,    0,  255,  153,     //bluegreen
  122,  200,  200,  200,     //gray
  181,  110,   61,    6,     //brown
  255,    6,  255,    0      //green
};
CRGBPalette16 brownGreenPalette = browngreen_gp;


DEFINE_GRADIENT_PALETTE( heatmap_gp ) {
  0,      0,    0,    0,  // black
  128,  255,    0,    0,  // red
  200,  255,  255,    0,  // bright yellow
  255,  255,  255,  255,  // full white
};
CRGBPalette16 heatPalette = heatmap_gp;

DEFINE_GRADIENT_PALETTE( greenblue_gp ) {
  0,   0,  255, 245,
  46,  0,  21,  255,
  179, 12, 250, 0,
  255, 0,  255, 245
};
CRGBPalette16 greenbluePalette = greenblue_gp;

DEFINE_GRADIENT_PALETTE( Sunset_Real_gp ) {
  0, 120,  0,  0,
  22, 179, 22,  0,
  51, 255, 104,  0,
  85, 167, 22, 18,
  135, 100,  0, 103,
  198,  16,  0, 130,
  255,   0,  0, 160
};
CRGBPalette16 sunsetPalette = Sunset_Real_gp;

CRGBPalette16 purplePalette = CRGBPalette16 (
                                CRGB::DarkViolet,
                                CRGB::DarkViolet,
                                CRGB::DarkViolet,
                                CRGB::DarkViolet,

                                CRGB::Magenta,
                                CRGB::Magenta,
                                CRGB::Linen,
                                CRGB::Linen,
```

```
                                CRGB::Magenta,
                                CRGB::Magenta,
                                CRGB::DarkViolet,
                                CRGB::DarkViolet,

                                CRGB::DarkViolet,
                                CRGB::DarkViolet,
                                CRGB::Linen,
                                CRGB::Linen
                               );

DEFINE_GRADIENT_PALETTE( fireandice_gp ) {
  0,  80,  2,  1,
  51, 206, 15,  1,
  101, 242, 34,  1,
  153,  16, 67, 128,
  204,   2, 21, 69,
  255,   1,  2,  4
};
CRGBPalette16 fireandicePalette = fireandice_gp;

DEFINE_GRADIENT_PALETTE( bhw2_turq_gp ) {
  0,   1, 33, 95,
  38,   1, 107, 37,
  76,  42, 255, 45,
  127, 255, 255, 45,
  178,  42, 255, 45,
  216,   1, 107, 37,
  255,   1, 33, 95
};
CRGBPalette16 turqPalette = bhw2_turq_gp;

DEFINE_GRADIENT_PALETTE( autumnrose_gp ) {
  0,  71,  3,  1,
  45, 128,  5,  2,
  84, 186, 11,  3,
  127, 215, 27,  8,
  153, 224, 69, 13,
  188, 229, 84,  6,
  226, 242, 135, 17,
  255, 247, 161, 79
};
CRGBPalette16 autumnrosePalette = autumnrose_gp;

DEFINE_GRADIENT_PALETTE( bhw1_06_gp ) {
  0, 184,  1, 128,
  160,   1, 193, 182,
```

```
  219, 153, 227, 190,
  255, 255, 255, 255
};
CRGBPalette16 bhw1_06Palette = bhw1_06_gp;

DEFINE_GRADIENT_PALETTE( bhw2_xmas_gp ) {
  0,   0, 12,  0,
  40,   0, 55,  0,
  66,   1, 117,  2,
  77,   1, 84,  1,
  81,   0, 55,  0,
  119,   0, 12,  0,
  153,  42,  0,  0,
  181, 121,  0,  0,
  204, 255, 12,  8,
  224, 121,  0,  0,
  244,  42,  0,  0,
  255,  42,  0,  0
};
CRGBPalette16 xmasPalette = bhw2_xmas_gp;

DEFINE_GRADIENT_PALETTE( bhw1_justducky_gp ) {
  0,  47, 28,  2,
  76, 229, 73,  1,
  163, 255, 255,  0,
  255, 229, 73,  1
};
CRGBPalette16 justduckyPalette = bhw1_justducky_gp;
```

Bibliografia

https://www.printables.com/model/146410-airsoft-arduino-bomb-replica-kajaki-airsoft

https://github.com/yinbot/Airsoft-BombPro

https://github.com/JorgeFilipePinto/Airsoft_Bomb_Montain_Wolves_Team.git


Politécnico
de Viseu
Tecnologia
e Gestão Lamego