Projeto - Expressor de emoções

Apresentação do projeto

O Expressor de Emoções é um dispositivo interativo projetado para auxiliar crianças com dificuldades de comunicação a expressarem seus sentimentos por meio de cores e sons. O projeto utiliza a placa BitDogLab e seu display OLED, LEDs RGB, Buzzers e Botões, permitindo que a criança selecione diferentes emoções e as represente de forma visual e auditiva.

Este projeto foi desenvolvido por John Kauã Gonçalves Lima, como parte da capacitação em Sistemas Embarcados feita pela equipe EmbarcaTech.

Título do projeto

Expressor de Emoções: Uma Interface de Comunicação para Crianças

Objetivos do projeto

- Criar um meio acessível e intuitivo para crianças com dificuldades de comunicação expressarem suas emoções.
- Utilizar elementos visuais (cores) e auditivos (sons) para representar sentimentos como alegria, tristeza, raiva e medo.
- Desenvolver um sistema interativo e responsivo que possa ser facilmente utilizado por crianças pequenas ou com necessidades especiais.
- Explorar tecnologias embarcadas para criar uma solução compacta e eficiente.

Descrição do funcionamento

O programa funciona da seguinte forma:

1. A criança interage com o sistema através de dois botões (navegação e confirmação).
2. O display OLED apresenta um menu com diferentes emoções, cada uma associada a uma cor, uma figura e um som específico.
3. A criança pode navegar pelo menu e selecionar uma emoção.
4. Durante a navegação um LED RGB muda de cor de cor de acordo com a emoção no display.

5. Ao confirmar a seleção, a matriz de LEDs RGB exibe a figura correspondente à emoção escolhida e um efeito sonoro é reproduzido.
6. O programa é redefinido para permitir uma nova seleção.

Justificativa

A dificuldade de expressão emocional pode ser um grande desafio para crianças com distúrbios de comunicação, como aquelas no espectro autista. O Expressor de Emoções oferece um meio intuitivo e acessível para que essas crianças possam demonstrar seus sentimentos, facilitando a interação com pais, cuidadores e terapeutas. Além disso, o projeto propõe uma solução tecnológica inovadora e de baixo custo, contribuindo para a inclusão social e o desenvolvimento emocional infantil.

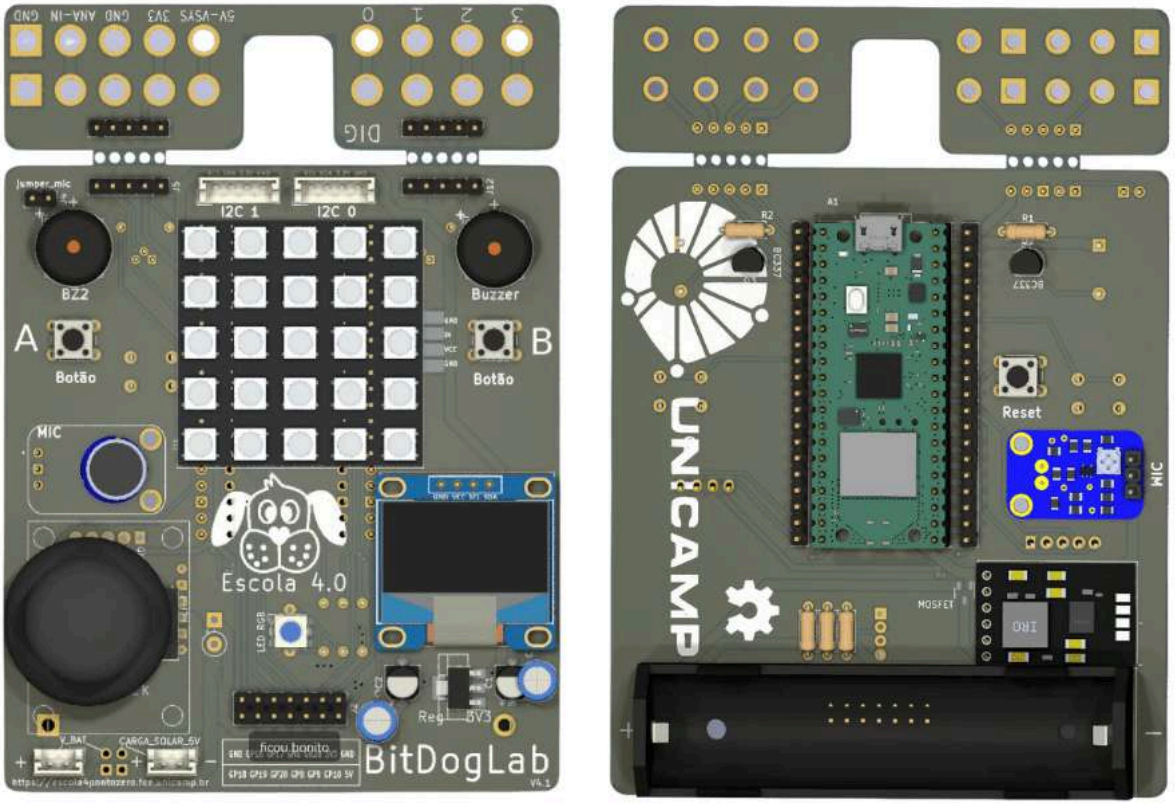
Originalidade

Projetos semelhantes ou correlatos:

- Emotismo
- Brinquedos Tang Toys
- Michelzinho

Especificação de hardware

Todo o hardware utilizado no projeto está na placa BitDoglab.



Repositório público da BitDogLab

Componentes utilizados

- 1 buzzer
  - Emite um sound effect de navegação e toca uma música relacionada a uma emoção.
- 2 botões
  - O botão B é responsável pela navegação, ao pressionar, a emoção selecionada muda. O botão A é responsável pela confirmação e ativa a figura desenhada na matriz de LEDs e a música reproduzida pelo buzzer.
- Display OLED
  - Exibe o menu de navegação, mostrando a função de cada botão e a emoção selecionada.
- LED RGB
  - Muda sua cor de acordo com a emoção selecionada, sendo um indicativo visual além da tela OLED.
- Matriz de LEDs
  - Quando o botão de confirmação é pressionado, exibe uma figura que representa a emoção escolhida.

Configurações

Resumo das Configurações

Componentes	**Setup**
GPIOs (LEDs, Buzzer e Botões)	Configura pinos e resistores pull-up
I2C (OLED SSD1306)	Inicialização do barramento e comunicação
PWM (Buzzer)	Configura geração de sons
PIO (Matriz de LEDs RGB)	Controle da matriz via WS2818b.
Interrupções (Botões)	Configuração para detecção de sinal

Inicialização do Hardware

No setup(), os pinos dos botões e LEDs são inicializados:

```
// função de setup para os componente utilizados
void setup(ssd1306_t *oled)
{
    // I2C Initialisation. Using it at 400Khz.
    i2c_init(I2C_PORT, 400 * 1000);

    gpio_set_function(I2C_SDA, GPIO_FUNC_I2C);
    gpio_set_function(I2C_SCL, GPIO_FUNC_I2C);
    gpio_pull_up(I2C_SDA);
    gpio_pull_up(I2C_SCL);

    // setup leds
    gpio_init(PIN_LED_RED);
    gpio_init(PIN_LED_GREEN);
    gpio_init(PIN_LED_BLUE);
    gpio_set_dir(PIN_LED_RED, GPIO_OUT);
    gpio_set_dir(PIN_LED_GREEN, GPIO_OUT);
    gpio_set_dir(PIN_LED_BLUE, GPIO_OUT);

    // setup botoes
    gpio_init(PIN_BUTTON_NEXT);
    gpio_set_dir(PIN_BUTTON_NEXT, GPIO_IN);
    gpio_pull_up(PIN_BUTTON_NEXT);

    gpio_init(PIN_BUTTON_CONF);
    gpio_set_dir(PIN_BUTTON_CONF, GPIO_IN);
    gpio_pull_up(PIN_BUTTON_CONF);

    // setup oled
    ssd1306_init(oled, 128, 64, 0x3c, I2C_PORT);
}
```

I2C: Inicializa a comunicação I2C a 400 kHz e configura os pinos SDA e SCL.
LEDs RGB: Define os pinos como saída.
Botões: São configurados como entrada com resistores pull-up internos.
Display OLED: Configura e inicializa o display SSD1306.

Inicialização da Matriz de LEDs RGB (Neopixel)

A matriz de LEDs RGB é controlada pelo PIO (Programmable I/O) do RP2040.

```
void npInit(uint pin)
{
    // Adiciona o programa PIO compilado para controle dos LEDs
    uint offset = pio_add_program(pio0, &ws2818b_program);
    np_pio = pio0;

    // Aloca uma máquina de estado PIO para o controle dos LEDs
    sm = pio_claim_unused_sm(np_pio, false);
    if (sm < 0)
    {
        np_pio = pio1;
        sm = pio_claim_unused_sm(np_pio, true); // Se nenhuma máquina
estiver livre, pânico
    }

    // Configura o PIO para o protocolo WS2818b (Neopixel)
    ws2818b_program_init(np_pio, sm, offset, pin, 800000.f);
}
```

Adiciona um programa PIO para controlar os LEDs RGB.
Aloca uma máquina de estado no PIO para gerenciar os LEDs.
Configura a máquina de estado para operar com LEDs WS2818b.

Configuração do PWM para Controle do Buzzer

O PWM é usado para gerar sons no buzzer, alterando a frequência das notas musicais.

```
void setup_pwm(uint slice, uint32_t period, uint pin)
{
    pwm_set_clkdiv(slice, DIVIDER_PWM);
    pwm_set_wrap(slice, period);
    pwm_set_gpio_level(pin, period / 3.3);
    pwm_set_enabled(slice, true);
}
```

Define o divisor do clock para o PWM.
Define o período do PWM com base na frequência desejada.
Ajusta a razão cíclica do sinal PWM.
Ativa o PWM.

Configuração das Interrupções dos Botões

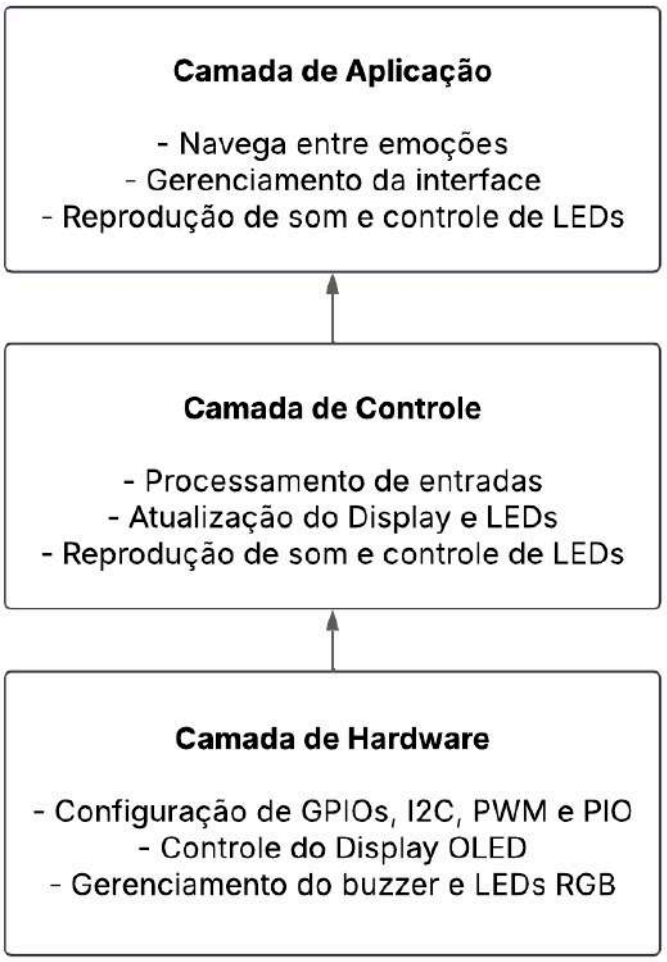
As interrupções são configuradas para detectar pressionamentos nos botões.

```
gpio_set_irq_enabled(PIN_BUTTON_CONF, GPIO_IRQ_EDGE_FALL, true);
gpio_set_irq_enabled_with_callback(PIN_BUTTON_NEXT, GPIO_IRQ_EDGE_FALL, true,
&button_callback);
```

Configura a interrupção para ativar quando o botão for pressionado.
Associa a interrupção ao button_callback().

Especificação do firmware

Diagrama de camadas do software



Funções

npWrite()

Função responsável por atualizar a matriz de LEDs RGB com a figura correspondente à emoção selecionada.

```
void npWrite()
{
    const npLED_t *leds = faces[current_emotion];

    // Escreve cada dado de 8 bits dos pixels no buffer da máquina PIO
    for (uint i = 0; i < LED_COUNT; ++i)
    {
        pio_sm_put_blocking(np_pio, sm, leds[i].G);
        pio_sm_put_blocking(np_pio, sm, leds[i].R);
        pio_sm_put_blocking(np_pio, sm, leds[i].B);
    }
    sleep_us(100); // Espera 100µs para garantir um reset conforme
especificado no datasheet.
}
```

1. Obtém os dados da matriz de LEDs referente à emoção atual (faces[current_emotion]).
2. Envia os valores RGB para cada LED na matriz utilizando PIO.
3. Aguarda um pequeno tempo (100 µs) para garantir que a atualização foi concluída corretamente.

npClear()

Função responsável por apagar os leds na matriz de LEDs

```
void npClear()
{
    // Envia zeros para todos os LEDs da matriz
    for (uint i = 0; i < LED_COUNT; ++i)
    {
        pio_sm_put_blocking(np_pio, sm, 0);
        pio_sm_put_blocking(np_pio, sm, 0);
        pio_sm_put_blocking(np_pio, sm, 0);
    }
    sleep_us(100); // Espera 100µs para garantir um reset.
}
```

1. Envia valores 0, 0, 0 (preto) para todos os LEDs, apagando a matriz.
2. Garante que os LEDs não fiquem com brilho residual ao desligá-los.

OBS.: As funções e estruturas de dados utilizadas para a utilização da matriz de LEDs RGB são de um exemplo público no github da BitDogLab-C: https://github.com/BitDogLab/BitDogLab-C/tree/main/neopixel_pio

set_led_color()

Função responsável por atualizar o LED RGB único do placa.

```
void set_led_color(bool red, bool green, bool blue)
{
    gpio_put(PIN_LED_RED, red);
    gpio_put(PIN_LED_GREEN, green);
    gpio_put(PIN_LED_BLUE, blue);
}
```

1. Define os níveis dos pinos de saída dos LEDs RGB (PIN_LED_RED, PIN_LED_GREEN, PIN_LED_BLUE).
2. Dependendo da combinação de true ou false, diferentes cores podem ser geradas.

display_emotion()

Função que atualiza a interface gráfica do display OLED, exibindo o menu de navegação.

```
void display_emotion(ssd1306_t *oled, const char *emotion)
{
    ssd1306_clear(oled);
    ssd1306_draw_string(oled, 0, 0, 1, "Escolha uma emocao:");
    ssd1306_draw_string(oled, 15, 18, 2, emotion);
    ssd1306_draw_string(oled, 0, 44, 1, "B - Proximo");
    ssd1306_draw_string(oled, 0, 54, 1, "A - Confirmar");
    ssd1306_show(oled);
}
```

1. Limpa a tela OLED antes de desenhar a nova interface.
2. Exibe:
    a. O título "Escolha uma emoção".
    b. O nome da emoção atual (emotion).
    c. Instruções sobre os botões (B - Próximo, A - Confirmar).
3. Atualiza a tela com ssd1306_show(oled).

OBS.: Para facilitar a utilização do display OLED, foi utilizado uma biblioteca pública no github: https://github.com/daschr/pico-ssd1306

play_emotion_song()

Função que toca músicas com o buzzer e desenha figuras na matriz de LEDs.

```
void play_emotion_song(uint slice)
{
    const Note *song = songs[current_emotion];
    uint song_length = song_lengths[current_emotion];

    npWrite(); // Atualiza os LEDs da matriz com a figura correspondente

    // Reproduz a música nota por nota
    for (uint i = 0; i < song_length; i++)
    {
        setup_pwm(slice, song[i].period, PIN_BUZZER_NAV);
        sleep_ms(song[i].duration);
    }

    npClear(); // Apaga os LEDs ao final da música

    // Desliga o som
    pwm_set_enabled(slice, false);
    pwm_set_gpio_level(PIN_BUZZER_NAV, 0);
    play_music = false;
}
```

1. Obtém a música associada à emoção atual.
2. Atualiza os LEDs RGB usando npWrite().
3. Percorre cada nota musical, ajustando a frequência do PWM do buzzer.
4. Desliga o PWM para silenciar o buzzer.
5. Apaga os LEDs com npClear() ao final da música.

button_callback()

Função callback ativada pelos botões A e B

```
void button_callback(uint gpio, uint32_t events)
{
    uint64_t current_time = to_ms_since_boot(get_absolute_time());

    if (gpio == PIN_BUTTON_NEXT) // Botão de navegação
    {
        if (current_time - last_button_press_nav > debounce_time_ms)
        {
            last_button_press_nav = current_time;
            current_emotion = (current_emotion + 1) % num_emotions;

            switch (current_emotion)
            {
            case 0: set_led_color(true, true, false); break; // Amarelo
            case 1: set_led_color(false, false, true); break; // Azul
            case 2: set_led_color(true, false, false); break; //
Vermelho
            case 3: set_led_color(false, true, false); break; // Verde
            }

            play_sfx = true;
            start_time_sfx = current_time;
        }
    }
    else if (gpio == PIN_BUTTON_CONF) // Botão de confirmação
    {
        if (current_time - last_button_press_conf > debounce_time_ms)
        {
            last_button_press_conf = current_time;
            play_music = true;
        }
    }
}
```

1. Verifica qual botão ativou a função
    a. Botão de navegação:
        i. “Anda no array de emoções”
        ii. Atualiza a cor do LED RGB
        iii. Ativa a flag para tocar o sound effect de navegação
    b. Botão de confirmação:
        i. Ativa a flag para tocar a música e desenhar a figura na matriz de LEDs.

Em ambos os casos há uma implementação de debouncing via software baseado em tempo, onde há um “cooldown time” para que os comandos sejam executados novamente.

main()

Função executada primeiro no programa e principal âncora de funcionalidades.

```
int main()
{
    stdio_init_all();
    npInit(PIN_LED_ARRAY);

    gpio_set_irq_enabled(PIN_BUTTON_CONF, GPIO_IRQ_EDGE_FALL, true);
    gpio_set_irq_enabled_with_callback(PIN_BUTTON_NEXT, GPIO_IRQ_EDGE_FALL,
true, &button_callback);

    gpio_set_function(PIN_BUZZER_NAV, GPIO_FUNC_PWM);
    uint slice = pwm_gpio_to_slice_num(PIN_BUZZER_NAV);

    ssd1306_t oled;
    oled.external_vcc = false;
    setup(&oled);

    set_led_color(true, true, false); // Inicia em amarelo

    while (true)
    {
        display_emotion(&oled, emotions[current_emotion]);

        if (play_sfx)
        {
            setup_pwm(slice, base_period, PIN_BUZZER_NAV);
            if (to_ms_since_boot(get_absolute_time()) - start_time_sfx >=
nav_sfx_duration)
            {
                play_sfx = false;
                pwm_set_enabled(slice, false);
            }
        }

        if (play_music)
        {
            play_emotion_song(slice);
        }

        tight_loop_contents();
    }
}
```

- Configura o hardware e inicia o sistema.
- Exibe a emoção atual no display (inicialmente: “Feliz”).
- Toca som e música quando necessário.
- Mantém o programa rodando continuamente.

OBS.: As funções setup(), setup_pwm() e npInit() já foram apresentadas.

Variáveis

Defines

```
// defines para pinos ou globais específicas
#define I2C_PORT i2c1           // Porta I2C
#define I2C_SDA 14              // I2C SDA
#define I2C_SCL 15              // I2C SCL
#define PIN_BUTTON_NEXT 6       // Pino botão de navegação
#define PIN_BUTTON_CONF 5       // Pino botão de confirmação
#define PIN_LED_RED 13          // Pino led vermelho
#define PIN_LED_BLUE 12         // Pino led verde
#define PIN_LED_GREEN 11        // Pino led azul
#define PIN_BUZZER_NAV 21       // Pino buzzer do sfx de navegação
#define DIVIDER_PWM 16.0        // Divisor de clock para o pwm
#define PIN_LED_ARRAY 7         // Pino matriz de leds RGB
#define LED_COUNT 25            // Quantidade de leds na matriz
```

Structs

```
// struct para notas musicais
typedef struct
{
    uint16_t period;   // frequência da nota
    uint64_t duration; // duração da nota
} Note;

// Definição de pixel GRB
struct pixel_t
{
    uint8_t G, R, B; // Três valores de 8 bits compõem um pixel.
};
typedef struct pixel_t pixel_t;
typedef pixel_t npLED_t; // Mudança de nome de "struct pixel_t" para
"npLED_t" por clareza.
```

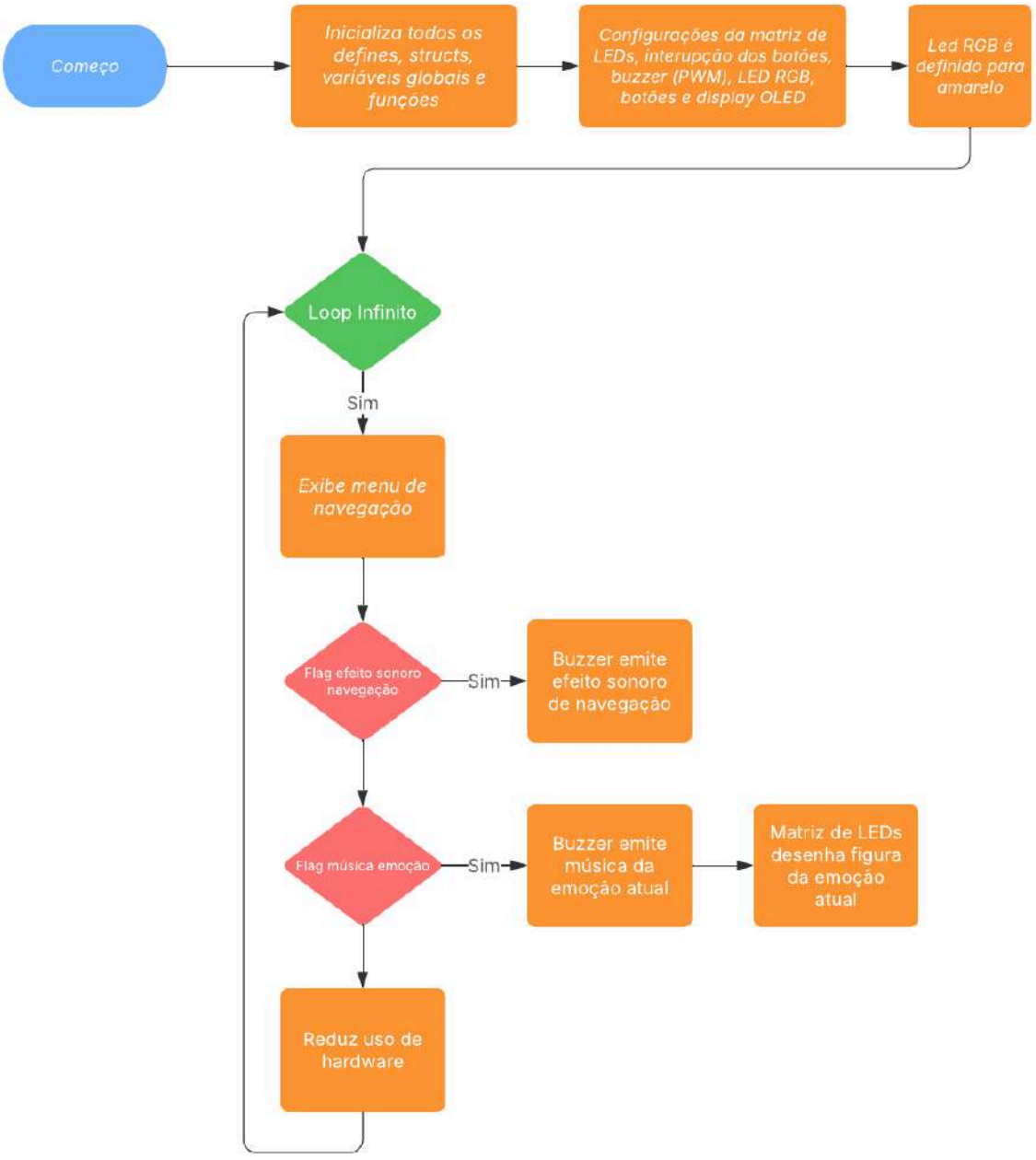
Variáveis Globais

```
//                                GLOBAIS

// músicas para cada emoção

const Note happy_song[] = {
    {1911, 150}, {1703, 150}, {1517, 150}, {1432, 150}, {1276, 300}};

const Note sad_song[] = {
    {1276, 300}, {1432, 300}, {1517, 300}, {1703, 500}};

const Note angry_song[] = {
    {1000, 100}, {900, 100}, {800, 100}, {700, 100}, {600, 200}};

const Note calm_song[] = {
    {1911, 500}, {1703, 500}, {1432, 500}, {1276, 700}};

// array com os tamanhos das músicas
const uint song_lengths[] = {
    sizeof(happy_song) / sizeof(happy_song[0]),
    sizeof(sad_song) / sizeof(sad_song[0]),
    sizeof(angry_song) / sizeof(angry_song[0]),
    sizeof(calm_song) / sizeof(calm_song[0])};

// Array de ponteiros para as músicas
const Note *songs[] = {happy_song, sad_song, angry_song, calm_song};

// arrays de leds para cada emoção
npLED_t happy_face[] = {
    {0, 0, 0}, {80, 80, 0}, {80, 80, 0}, {80, 80, 0}, {0, 0, 0}, {80,
80, 0}, {80, 80, 0}, {80, 80, 0}, {80, 80, 0}, {80, 80, 0}, {0, 0, 0},
{0, 0, 0}, {0, 0, 0}, {0, 0, 0}, {0, 0, 0}, {0, 0, 0}, {80, 80, 0}, {0,
0, 0}, {80, 80, 0}, {0, 0, 0}, {0, 0, 0}, {80, 80, 0}, {0, 0, 0}, {80,
80, 0}, {0, 0, 0}};

npLED_t sad_face[] = {
    {0, 0, 0}, {0, 0, 80}, {0, 0, 0}, {0, 0, 80}, {0, 0, 0}, {0, 0, 0},
{0, 0, 80}, {0, 0, 80}, {0, 0, 80}, {0, 0, 0}, {0, 0, 0}, {0, 0, 0}, {0,
0, 0}, {0, 0, 0}, {0, 0, 0}, {0, 0, 80}, {0, 0, 80}, {0, 0, 0}, {0, 0,
80}, {0, 0, 80}, {0, 0, 0}, {0, 0, 80}, {0, 0, 0}, {0, 0, 80}, {0, 0,
0}};

npLED_t angry_face[] = {
    {0, 80, 0}, {0, 80, 0}, {0, 80, 0}, {0, 80, 0}, {0, 80, 0}, {0, 0,
```

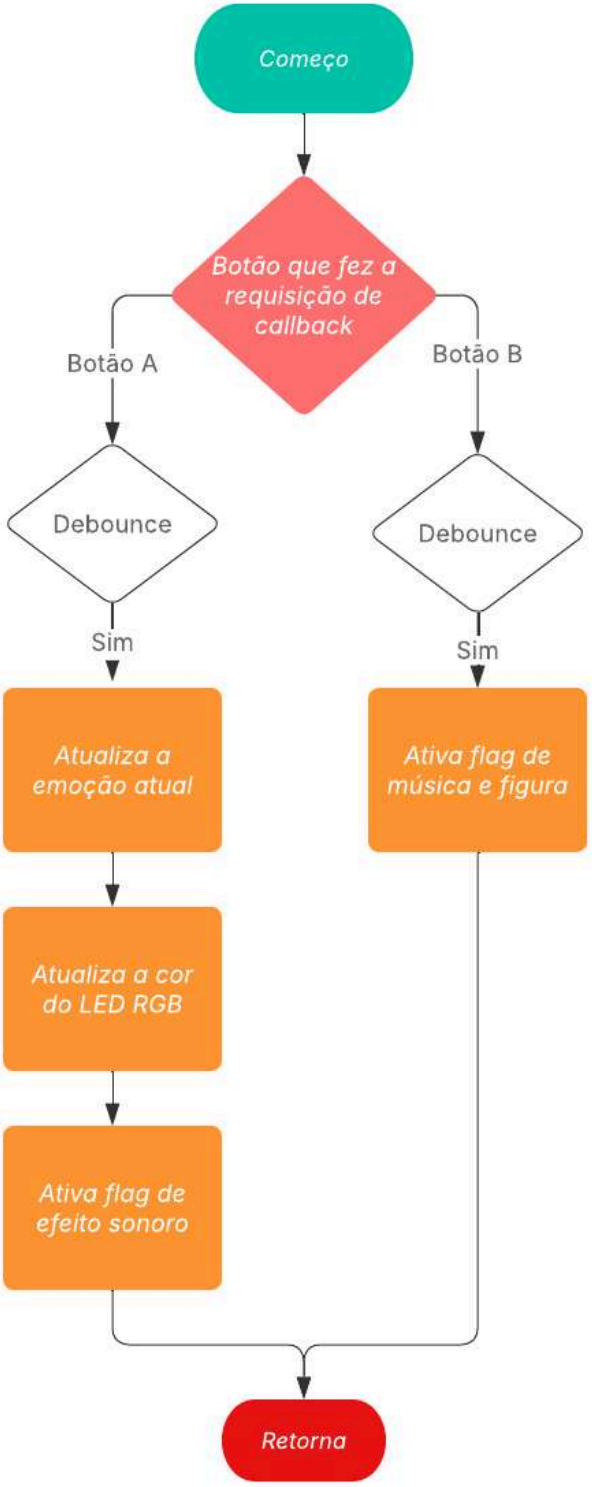
```
0}, {0, 80, 0}, {0, 80, 0}, {0, 80, 0}, {0, 0, 0}, {0, 0, 0}, {0, 0, 0},
{0, 0, 0}, {0, 0, 0}, {0, 0, 0}, {0, 0, 0}, {0, 80, 0}, {0, 0, 0}, {0,
80, 0}, {0, 0, 0}, {0, 80, 0}, {0, 0, 0}, {0, 0, 0}, {0, 0, 0}, {0, 80,
0}};

npLED_t calm_face[] = {
    {0, 0, 0}, {80, 0, 0}, {80, 0, 0}, {80, 0, 0}, {0, 0, 0}, {80, 0,
0}, {0, 0, 0}, {0, 0, 0}, {0, 0, 0}, {80, 0, 0}, {0, 0, 0}, {0, 0, 0},
{0, 0, 0}, {0, 0, 0}, {0, 0, 0}, {0, 0, 0}, {80, 0, 0}, {0, 0, 0}, {80,
0, 0}, {0, 0, 0}, {0, 0, 0}, {80, 0, 0}, {0, 0, 0}, {80, 0, 0}, {0, 0,
0}};

// Array de ponteiros para os desenhos
const npLED_t *faces[] = {happy_face, sad_face, angry_face, calm_face};

const uint base_period = 2000; // Período base do pwm
const uint period = base_period; // Periodo do buzzer
volatile bool play_sfx = false; // Flag para indicar se o som deve tocar
volatile bool play_music = false; // Flag para indicar se a música deve
tocar
volatile uint64_t start_time_sfx = 0; // Marca o início do som
const uint64_t nav_sfx_duration = 100; // Duração do som em
milissegundos
volatile uint64_t last_button_press_nav = 0; // Marca o último evento do
botão de navegação
volatile uint64_t last_button_press_conf = 0; // Marca o último evento
do botão de confirmação
const uint64_t debounce_time_ms = 200; // Tempo de debounce (200ms)
const char *emotions[] = {"Feliz:D", "Triste:( ", "Bravo>:(",
"Calmo:)"}; // Array de emoções
const uint num_emotions = 4; // Tamanho do array de emoções
volatile uint current_emotion = 0; // Emoção atual na navegação

PIO np_pio; // representa um periférico PIO
uint sm; // armazena o número da máquina de estado do PIO.
```

Fluxogramas principais

main()



button_callback()



play_emotion_song()

Dados específicos

struct Nota

Atributos:
- Frequência
- Duração

Músicas

Arrays de structs Nota.

Figuras

Arrays de structs de valores RGB (fonte: exemplo neo_pixel repositório da BitDogLab).

Flags

1 para efeito sonoro de navegação e 1 para tocar música

Array de emoções

Utilizada para mapear a emoção atual da navegação. Rege maior parte do funcionamento do programa.

Auxiliares

Definem padrões de valores no código (durações, frequências, tempos limite e etc.)

Inicialização

Configuração da Comunicação Serial

```
stdio_init_all();
```

Permite que o Raspberry Pi Pico envie mensagens de depuração pela USB (útil para testes e logs).

Inicialização da Matriz de LEDs RGB

```
npInit(PIN_LED_ARRAY);
```

A função npInit() configura o Programmable I/O (PIO) para gerar os sinais necessários para controlar os LEDs da matriz WS2818.

Configuração das Interrupções para os Botões

```
gpio_set_irq_enabled(PIN_BUTTON_CONF, GPIO_IRQ_EDGE_FALL, true);
gpio_set_irq_enabled_with_callback(PIN_BUTTON_NEXT, GPIO_IRQ_EDGE_FALL,
true, &button_callback);
```

Configura interrupções em ambos os botões e uma associa uma função callback a ambos os botões.

Configuração do PWM para o Buzzer

```
gpio_set_function(PIN_BUZZER_NAV, GPIO_FUNC_PWM);
uint slice = pwm_gpio_to_slice_num(PIN_BUZZER_NAV);
```

Configura o buzzer como saída PWM, permitindo gerar tons musicais.
Obtém o slice PWM associado ao pino do buzzer.

Inicialização do Display OLED

```
ssd1306_t oled;
oled.external_vcc = false;
setup(&oled);
```

Cria uma estrutura oled para representar a tela.
Chama setup(&oled), que:

- Inicializa a comunicação I2C com o display.
- Configura o tamanho e endereço I2C do OLED.

Define as configurações do display.

Definição da Cor Inicial do LED RGB

```
set_led_color(true, true, false); // Amarelo
```

Define o LED RGB para a cor amarela, que representa a emoção inicial.

Execução do projeto

O projeto foi inteiramente feito por uma pessoa, John Kauã Gonçalves Lima, desde sua concepção até implementação e documentação.

A IDE utilizada foi o Visual Studio Code, com auxílio de extensões para facilitar o

trabalho com a placa Raspberry Pi Pico W e trabalhos em CMake.

A documentação foi feita na plataforma Google Docs, como uso de ferramentas para desenho de diagramas (Lucid), extensão para formatação de blocos de código (Code Blocks).

Git e GitHub foram utilizados para controle de versionamento do projeto.

Metodologia

O projeto foi feito de maneira iterativa e incremental, com auxílio de Git e GitHub. Os testes foram feitos de maneira conjunta com a implementação, logo após a finalização de uma feature. Todos os teste foram rodados na BitDogLab disponibilizada pela equipe EmbarcaTech.

Bibliotecas e/ou códigos auxiliares utilizados:

https://github.com/daschr/pico-ssd1306
https://github.com/BitDogLab/BitDogLab-C/tree/main/neopixel_pio

Referências

GITHUB

DASCHR. *pico-ssd1306*. Disponível em: https://github.com/daschr/pico-ssd1306. Acesso em: 9 fev. 2025.

BITDOGLAB. *BitDogLab-C: neopixel_pio*. Disponível em: https://github.com/BitDogLab/BitDogLab-C/tree/main/neopixel_pio. Acesso em: 9 fev. 2025.

ARTIGOS ACADÊMICOS

PAGNONCELLI, W.; MÜLLER, T. Inteligência artificial e aprendizagem de máquina no ensino de programação: uma revisão sistemática da literatura. *Anais do Simpósio Brasileiro de Informática na Educação (SBIE)*, v. 31, 2020. Disponível em: https://sol.sbc.org.br/index.php/sbie/article/view/12825. Acesso em: 9 fev. 2025.

ZHU, X.; WU, X. Attention mechanisms in neural networks: a survey. *arXiv preprint*, 2020. Disponível em: https://arxiv.org/abs/2007.05286. Acesso em: 9 fev. 2025.

NOTÍCIAS

UNIVERSIDADE FEDERAL DE UBERLÂNDIA. Inteligência artificial auxilia autistas a identificar emoções. *Comunica UFU*, 22 ago. 2023. Disponível em: https://comunica.ufu.br/noticias/2023/08/inteligencia-artificial-auxilia-autistas-identificar-emocoes. Acesso em: 9 fev. 2025.