Projeto Final

Maria Elaine de Holanda Cavalcante
CPF: 080.528.073-10

Escopo do Projeto

Título: Blocos Coloridos

O projeto consiste de um jogo lúdico de blocos. A cada momento, um bloco colorido é gerado na sua matriz de LEDs, esse bloco está sempre caindo, mas você pode controlá-lo através de três botões: um para movimentar o bloco para a esquerda, outro para movimentar o bloco para direita, e mais outro para fazer o bloco cair mais rápido. Quando o bloco atinge o final da matriz de LEDs, ele permanece lá e um novo bloco colorido é gerado em seguida. O objetivo do jogo é agrupar blocos de mesma cor! Quando três ou mais blocos de cores iguais são postos ortogonalmente consecutivos, os blocos somem e geram pontos para você. Os blocos que estavam acima dos blocos que sumiram caem para preencher toda a parte de baixo da matriz de LEDs. Quando não há mais espaço para novos blocos, o jogo se encerra. O seu objetivo é pontuar o máximo possível antes que o espaço na matriz acabe!

Objetivos do projeto

O projeto tem um caráter lúdico, feito para divertir. É um projeto simples e cativante que pode ser utilizado para amostragem em escolas e oficinas com crianças e adolescentes, para atrair a atenção, aproximar o contato com a área de sistemas embarcados e despertar o interesse.

Funcionalidades

As funcionalidades do projeto incluem manipular um bloco exibido na matriz de LEDs utilizando três botões, um para movimentar para esquerda, outro para direita e outro para baixo. Inclui a geração pseudoaleatória de um novo bloco colorido e a exibição de padrões coloridos na matriz de LEDs de forma responsiva às ações do jogador.

Especificações do hardware

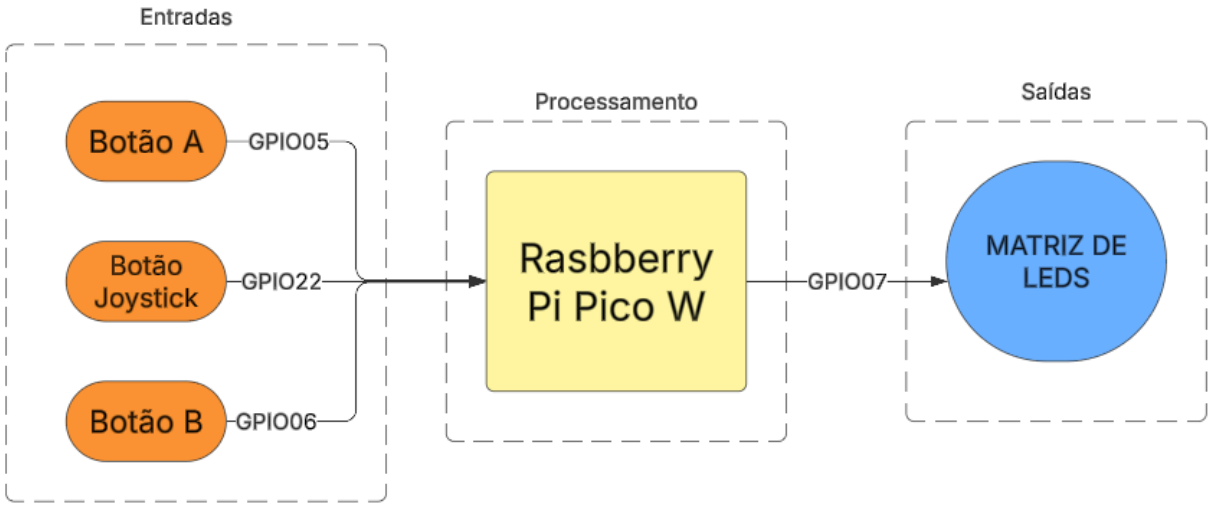
O projeto foi implementado na placa BitDogLab, mas os componentes de hardware necessários são somente:

- Raspberry Pi Pico W
- Matriz de LEDs 5x5
- 3 botões

Diagrama em blocos

A interação entre os componentes de hardware se dá da seguinte forma:

- **Grupo de Entrada**: composto pelos botões, que serão acionados de forma não periódica, de acordo com o usuário. Os botões foram configurados com resistores pull-up, os botões estão em estado baixo se estão sendo pressionados, e também foram configurados para gerar interrupções de borda de descida nas portas GPIO. Dessa forma, o Grupo de Processamento não fica a todo momento checando o estado dos botões, mas sim espera uma interrupção acontecer.
- **Grupo de Processamento**: composto unicamente pela Raspberry Pi Pico W, que fará a leitura dos botões quando uma interrupção das portas GPIO acontecer. A função do Grupo de Processamento é identificar qual botão foi acionado para então gerar uma resposta correspondente através da matriz de LEDs.
- **Grupo de Saída**: composto unicamente pela matriz de LEDs. A matriz recebe um buffer do Grupo de Processamento com os respectivos dados que se transcrevem em padrões de cores nos 25 LEDs da matriz e que devem ser mostrados ao usuário.



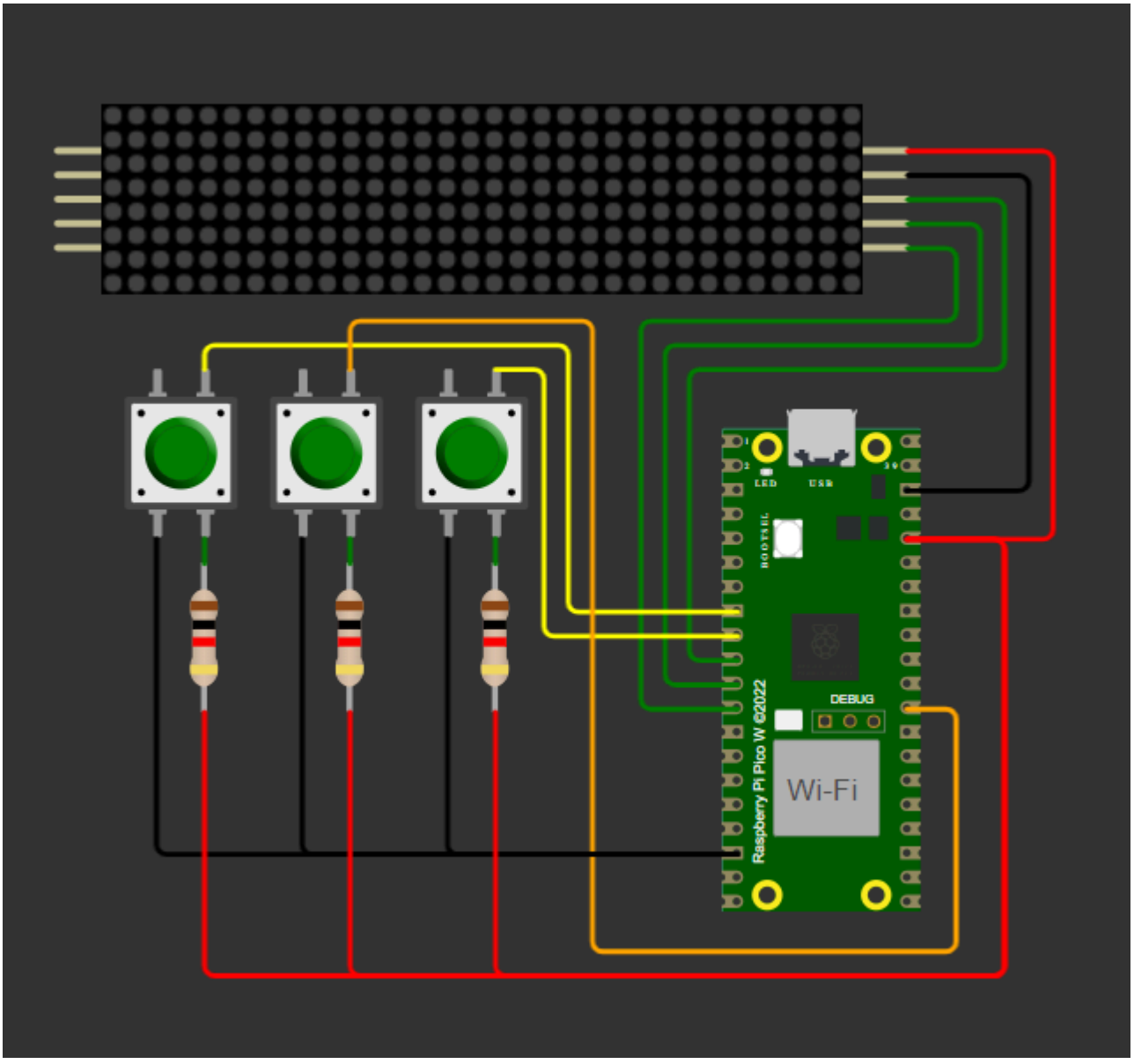
Pinagem

O botão A, cuja referência é a esquerda, está conectado à porta GPIO05 da Raspberry. O botão B, cuja referência é a direita, está conectado à porta GPIO06 da Raspberry. O botão do joystick está conectado à porta GPIO22 da Raspberry. As portas GPIO05, GPIO06 e GPIO22 são as que serão utilizadas pela Raspberry para leitura e recebimento do estado dos botões. Cada botão está também conectado ao GND e ao 3V3 da Raspberry, pinos responsáveis pela alimentação dos botões, e possuem resistores pull-up.
A entrada de dados da matriz de LEDs está conectada ao pino GPIO07 da Raspberry, este é o pino que será utilizado pela Raspberry para envio dos dados. Os pinos CS e CLK da matriz podem ser conectados à qualquer pino GPIO livre da Raspberry, embora seja uma boa prática conectar o pino CS da matriz a um pino da Raspberry que seja GPIO CS. Esses pinos são utilizados para a sincronização e comunicação da matriz com a Raspberry. Os

pinos GND e VCC são conectados aos pinos GND e 3V3 da Raspberry e são os pinos responsáveis pela alimentação da matriz.
A pinagem utilizada segue o padrão da placa BitDogLab.
As configurações utilizadas para a matriz de LEDs foram tiradas do repositório *neopixel_pio* do GitHub da BitDogLab (link nas referências deste documento).

Circuito

Sem a placa BitDogLab, utilizando apenas a Raspberry Pi Pico W, a matriz de LEDs e três botões, teremos o circuito abaixo. Vale lembrar que é necessária a biblioteca *ws2818b.pio.h* para o funcionamento correto da matriz. Essa biblioteca é disponibilizada no GitHub da BitDogLab, no repositório *neopixel_pio*, o link está presente nas referências deste documento.



Especificações do firmware

Blocos funcionais

O firmware do projeto só possui uma camada, pois não há comunicação com outros firmwares, o funcionamento inteiro do projeto, leitura de entradas, processamento e geração de saídas, acontece somente no código gravado na placa de uma Raspberry Pi Pico W.

Variáveis

As principais variáveis utilizadas são:
A matriz *led* de 7 linhas e 5 colunas, para representação do estado atual do jogo;
As flags de cada botão, para identificação de qual botão foi pressionado;
A matriz *componente*, que identifica quais blocos devem ser removidos;
A variável *tamanho_comp*, que guarda o tamanho da componente encontrada;
As variáveis *x* e *y*, que guardam a posição atual do bloco gerado.
Estas são as variáveis mais críticas ao funcionamento do firmware, mas ainda existem muitas outras que contribuem para a boa legibilidade do código e o funcionamento correto do projeto.

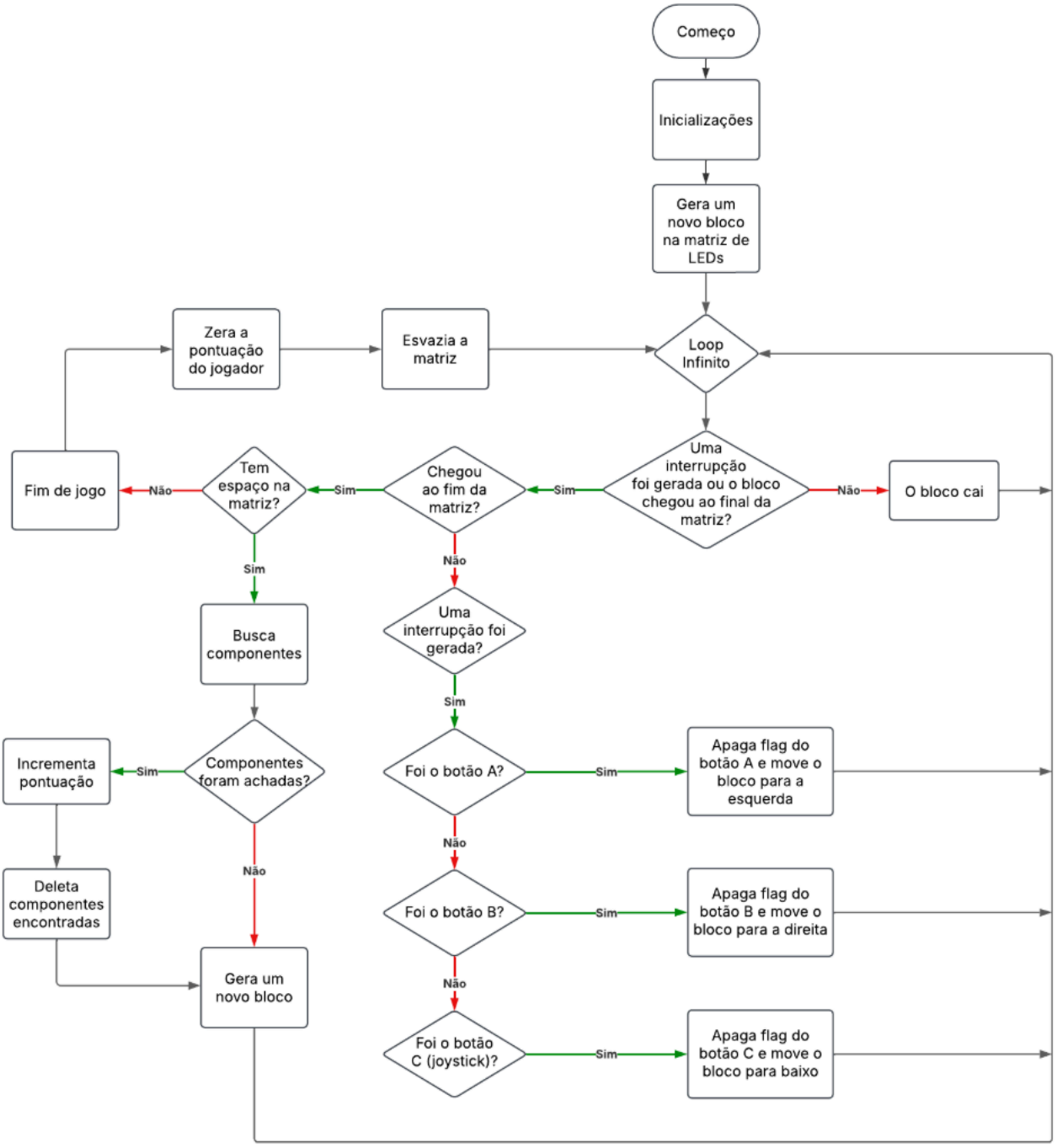
Funções

Breve explicação das principais funções do código:

- *setLEDColor* - seta a cor de um LED da matriz
- *drawColor* - seta a cor de um elemento da matriz *led*
- *writeColor* - seta a cor de todos os elementos da matriz
- *int_botao* - função que trata a interrupção GPIO gerada por algum dos botões
- *setup* - inicializações gerais e dos botões
- *gerarBloco* - gera um bloco de cor aleatória na linha zero e coluna 2 da matriz
- *paraBaixo* - move o bloco atual para baixo
- *paraDireita* - move o bloco atual para direita
- *paraEsquerda* - move o bloco atual para esquerda
- *buscar_componente* - encontra uma componente de blocos e armazena resultados na matriz *componente* e em *tamanho_comp*
- *reseta_componente* - reseta valores encontrados por buscar_componente
- *apaga_componente* - apaga componente que foi encontrada
- *varredura* - percorre toda a matriz e faz buscas de componentes nos elementos
- *tela_cheia* - retorna true se não há mais espaço vazio na matriz de LEDs
- *derrota* - animação de fim de jogo

Fluxograma

Esquema simplificado do funcionamento do firmware.



Inicializações e configurações

Os pinos dos botões são configurados com interrupções por borda de descida e com resistores pull-up.
A configuração da matriz de LEDs é feita de acordo com o repositório *neopixel_pio* do GitHub da BitDogLab (link nas referências).

Estruturas de Dados e Organização da Memória

As estruturas de dados utilizadas são somente inteiros. Buscou-se manter a padronização de todos os tipos de dados utilizados para evitar conflitos de tipagem.
A memória não é utilizada diretamente.

Execução do projeto

Metodologia

Escolha do projeto

Inicialmente, a escolha do projeto deveria ser capaz de ser lúdica, visualmente chamativa e divertida para cumprir com o objetivo de ser amostrada em escolas. Era necessário que fosse também interativa e cativante. Além disso, a escolha também precisava ser aplicável na placa BitDogLab, utilizando apenas os componentes nativos da placa, para também facilitar a futura reprodução do projeto. Dos componentes disponíveis na placa, a matriz de LEDs e o display OLED são os mais propícios para o desenvolvimento de jogos simples, então o projeto se voltou para eles. O display OLED não seria ideal devido ao seu tamanho muito pequeno, então optou-se pela matriz de LEDs para composição de um jogo. Dadas agora as limitações e recursos da matriz de LEDs, o jogo deveria ser composto por poucos quadrados e seria capaz de ter cores, a partir disso o joguinho de blocos foi escolhido como projeto final.

Pesquisas

Partindo daí, foram feitas pesquisas sobre a matriz de LEDs usando como principal referência o github da própria BitDogLab. Nele são oferecidos os arquivos, bibliotecas e o código inicial necessário para trabalhar sobre a matriz. Fora os botões usados para controlar o movimento dos blocos, a programação do projeto é voltada inteiramente para a manipulação dos LEDs da matriz, a matriz é constantemente atualizada para dar o efeito de animação do jogo, e possui diversas funções que foram implementadas para cada tipo de situação do jogo. A implementação partiu das seguintes etapas:

Teste do código inicial

Simplesmente testar o código já pronto do GitHub da BitDogLab. Para acessar esse código, basta baixar o repositório de nome *neopixel_pio* . O arquivo que será modificado para conter o código implementado é o de nome *neopixel_pio.c*. O código inicial permite que você imprima padrões estáticos de cores na matriz de LEDs.

Adaptação das funções

Muitas das funções pré-prontas utilizam um array unidimensional de 25 posições para se referir a cada LED individual da matriz, mas, para esse projeto, é mais interessante trabalhar com um array bidimensional de 35 posições, com 7 linhas e 5 colunas. As duas linhas adicionais são uma convenção para facilitar os processos: os blocos gerados são gerados em uma linha fora da matriz (linha 0), e o "final" da matriz é uma linha também fora da matriz (linha 6).

Novas funções

É feito um código de cores, o nosso array bidimensional conterá inteiros de 0 a 8, em que cada número corresponde a uma cor.
Além das funções adaptadas, são implementadas novas funções para fazer a conversão do código de cores criado para o código RGB que será passado aos LEDs, uma função para

gerar um novo bloco na linha 0 e coluna 2 da matriz *led*. A cor desse novo bloco é gerada de forma pseudoaleatória através da função *rand()*. Essa função utiliza uma seed para gerar a sequência pseudoaletória, é possível definir essa seed através do comando *srand()*, e podemos deixar a geração de blocos tão aleatória quanto possível ao passarmos a função *get_absolute_time()* como parâmetro de *srand()*, pois dessa forma garantimos que a cada vez que o jogo for reiniciado, a sequência gerada será diferente. Também foram implementadas funções para percorrer a matriz de LEDs e carregar todas as cores nos LEDs.

Primeiras animações

Em seguida, é necessário escrever as funções básicas de animação. Primeiro é programado o comportamento padrão do bloco: sempre cair. A dados momentos de tempo, fazemos com que o bloco passe para o LED de baixo, implementamos a função *paraBaixo()*. Em seguida, usando a mesma lógica, implementamos *paraEsquerda()* e *paraDireita()*.

Integração com botões

Agora que já temos as animações base, são inseridos os três botões. Na BitDogLab, o botão A, na porta 5, aciona a função *paraEsquerda()*. O botão B, na porta 6, aciona *paraDireita()*. O botão do joystick, na porta 22, aciona *paraBaixo()*, como o bloco já cai naturalmente, a função deste botão é fazê-lo cair mais rapidamente.
Os botões são lidos por meio de uma interrupção das portas GPIO da placa. Essa interrupção é gerada quando qualquer um dos botões gera uma borda de descida, e é tratada identificando qual dos botões gerou a interrupção e acionando a flag do respectivo botão. As flags dos botões são variáveis globais do tipo bool. No loop while(true) da função main, se uma interrupção é detectada, vemos qual flag está acionada, apagamos a flag e chamamos a função correspondente àquele botão.

Amostragem da pontuação

Mostrar o valor da pontuação do jogador, armazenada na variável *pontos* no monitor serial toda vez que ela for incrementada.

Identificação de componentes de blocos

Dois blocos estão na mesma componente se eles possuem a mesma cor e se existe um caminho ortogonal entre eles que só passa por blocos também da mesma cor.
Agora é necessário saber identificar quando existem blocos de cor igual juntos, apagar esses blocos e incrementar a pontuação do jogador. Para isso, é implementada uma função de varredura. Toda vez que um novo bloco é alojado na matriz, a função *varredura()* é chamada para percorrer a matriz em busca de componentes de blocos. Quando a matriz é percorrida, procuramos por elementos que tenham valor não-nulo, ou seja, aquele LED daquela posição da matriz não está apagado, e realizamos uma busca nesse LED. A função *buscar_componente()* consiste em percorrer os vizinhos de mesma cor do elemento, adicioná-los como elementos que estão na mesma componente, e então fazer a mesma busca em cada um dos vizinhos que possui mesma cor. Ao final do algoritmo, teremos salvo todos os LEDs que estão na componente em uma outra matriz *componente*, e teremos

salvo em uma variável *tamanho_comp* a quantidade de LEDs que possuem mesma cor, se essa quantidade for maior ou igual a 3, nós apagamos todos os LEDs da componente encontrada e incrementamos a pontuação do jogado em *tamanho_comp*. A função *apagar_componente()* irá percorrer a matriz, apagar todos os elementos que estiverem setados na matriz *componente* e irá mover para baixo todos os blocos que estiverem acima de algum bloco que foi apagado.
Mas a função de *varredura()* não para por aí, é possível que, após apagar uma componente, outras componentes tenham se formado, pois houveram blocos que foram movidos. Portanto, é preciso continuar percorrendo a matriz e fazendo buscas até que nenhuma componente seja achada.

Implementação do fim do jogo
Identificação do *GAME OVER*, que será quando algum bloco for alojado fora da matriz, ou seja, na linha 0, ou quando a matriz estiver inteiramente preenchida. Uma vez que um destes estados seja identificado, é gerada uma tela de derrota e a pontuação do jogador é zerada.

Testes de validação

Os testes feitos foram feitos a cada etapa de implementação e são observados a partir do experimento do próprio jogo: os blocos respondem ao movimento dos botões; a movimentação dos blocos é constante e sempre dentro da matriz; a exclusão de componentes de blocos acontece corretamente e uma cor branca é mostrada sobre toda a componente para facilitar a visualização e acrescentar à estética do jogo; a pontuação do jogador, mostrada no monitor serial, é incrementada corretamente; o fim do jogo acontece como esperado, quando não é mais possível inserir novos blocos na matriz.

Discussão dos resultados

Os resultados são satisfatórios, mas para cumprir com o objetivo principal do projeto, a amostragem em escolas, é necessário fazer alguns ajustes, como implementar uma forma de reduzir a luminosidade da matriz de LEDs, implementar uma visualização mais direta da pontuação do jogador, através de displays, implementar telas de menus de início e de fim de jogo, implementar efeitos sonoros de acordo com as ações do jogo e aumentar a suavidade das animações, além da substituição dos botões por um joystick, a fim de melhorar a jogabilidade.

Demonstração do projeto

Vídeo curto do projeto em funcionamento: https://www.youtube.com/watch?v=H3bIkeS04-M

Referências

Sobre a matriz de LEDs:
https://github.com/BitDogLab/BitDogLab-C/tree/main/neopixel_pio

https://github.com/BitDogLab/BitDogLab-C/blob/main/neopixel_pio/Readme.md
Sobre as interrupções:
Ebook Unidade 4 Capítulo 4 do Embarcatech.

Código Utilizado

Lembrando que, para funcionamento do código abaixo na placa BitDogLab, basta baixar o repositório *neopixel_pio* do GitHub da BitDogLab (link nas referências deste documento) e substituir o arquivo *neopixel_pio.c* pelo código abaixo.

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <stdbool.h>
#include <stdlib.h>
#include <time.h>
#include "pico/stdlib.h"
#include "pico/time.h"
#include "hardware/pio.h"
#include "hardware/gpio.h"
#include "hardware/timer.h"
#include "hardware/clocks.h"
#include "ws2818b.pio.h"

#define botao_a 5
#define botao_b 6
#define botao_c 22

//////////-------Configurações da matriz de LEDs--------------Linhas 20 a
73------/////////////
#define LED_COUNT 25
#define LED_PIN 7

struct pixel_t {
  uint8_t G, R, B;
};
typedef struct pixel_t pixel_t;
typedef pixel_t npLED_t;

npLED_t leds[LED_COUNT];

PIO np_pio;
uint sm;

void npInit(uint pin) {

  uint offset = pio_add_program(pio0, &ws2818b_program);
  np_pio = pio0;
```

```
  sm = pio_claim_unused_sm(np_pio, false);
  if (sm < 0) {
    np_pio = pio1;
    sm = pio_claim_unused_sm(np_pio, true);
  }

  ws2818b_program_init(np_pio, sm, offset, pin, 800000.f);

  for (uint i = 0; i < LED_COUNT; ++i) {
    leds[i].R = 0;
    leds[i].G = 0;
    leds[i].B = 0;
  }
}

void npSetLED(const int index, const uint8_t r, const uint8_t g, const
uint8_t b) {
  leds[index].R = r;
  leds[index].G = g;
  leds[index].B = b;
}

void npClear() {
  for (int i = 0; i < LED_COUNT; ++i)
    npSetLED(i, 0, 0, 0);
}

void npWrite() {
  for (int 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);
  return;
}
///////////-------Configurações da matriz de LEDs---------------Linhas 20 a
73------//////////////

volatile int ind[7][5]={  //matriz para converter a matriz led[7][5] para o
buffer
  {-1, -1, -1, -1, -1},
  {24, 23, 22, 21, 20},
  {15, 16, 17, 18, 19},
  {14, 13, 12, 11, 10},
  {5, 6, 7, 8, 9},
  {4, 3, 2, 1, 0},
  {-1, -1, -1, -1, -1}
};
```

```
volatile int led[7][5]={  //matriz que representa o estado atual do jogo
  {0, 0, 0, 0, 0},
  {0, 0, 0, 0, 0},
  {0, 0, 0, 0, 0},
  {0, 0, 0, 0, 0},
  {0, 0, 0, 0, 0},
  {0, 0, 0, 0, 0},
  {11, 11, 11, 11, 11}
};

// 0 apagado
// 1 vermelho
// 2 verde
// 3 azul
// 4 amarelo
// 5 rosa
// 6 roxo
// 7 laranja
// 8 branco

void setLEDColor(int i, int j, int r, int g, int b){  //seta a cor de um LED
da matriz
  int index = ind[i][j];
  npSetLED(index, r, g, b);
  return;
}

void drawColor(int i, int j){                         //seta a cor de um
elemento da matriz
  const uint c=led[i][j];
  if(c==0) setLEDColor(i, j, 0, 0, 0);                // apagado
  else if(c==1) setLEDColor(i, j, 50, 0, 0);          // vermelho
  else if(c==2) setLEDColor(i, j, 0, 50, 0);          // verde
  else if(c==3) setLEDColor(i, j, 0, 0, 50);          // azul
  else if(c==4) setLEDColor(i, j, 130, 120, 0);       // amarelo
  else if(c==5) setLEDColor(i, j, 127, 20, 20);       // rosa
  else if(c==6) setLEDColor(i, j, 50, 0, 50);         // roxo
  else if(c==7) setLEDColor(i, j, 127, 20, 0);        // laranja
  else if(c==8) setLEDColor(i, j, 127, 127, 127);     //branco
  else printf("cor invalida\n");
  return;
}

void writeColor(){                                    //seta a cor de todos
os elementos da matriz
  for(int i=1; i<6; i++){
    for(int j=0; j<5; j++) drawColor(i, j);
  }
  npWrite();
```

```
  sleep_ms(10);
  return;
}

volatile int x, y;                //coordenadas do bloco atual
volatile int speed=500;           //velocidade do jogo
int pontos=0;                     //pontuação
volatile bool piso=false;           //variável que indica se chegou o fim da
matriz
volatile bool intf=false;           //flag interrupção GPIO
volatile bool flagBotao_a=false;    //flags dos botões
volatile bool flagBotao_b=false;
volatile bool flagBotao_c=false;

void int_botao(uint gpio, uint32_t events){     //tratamento da interrupção
dos botões
  intf=true;
  if(gpio==botao_a) flagBotao_a=true;
  if(gpio==botao_b) flagBotao_b=true;
  if(gpio==botao_c) flagBotao_c=true;
  return;
}

void setup(){                         //inicializações gerais e dos botões
  stdio_init_all();

  gpio_init(botao_a);                 //configurações botao a
  gpio_set_dir(botao_a, GPIO_IN);
  gpio_pull_up(botao_a);

  gpio_init(botao_b);                 //configurações botao b
  gpio_set_dir(botao_b, GPIO_IN);
  gpio_pull_up(botao_b);

  gpio_init(botao_c);                 //configurações botao c
  gpio_set_dir(botao_c, GPIO_IN);
  gpio_pull_up(botao_c);

  gpio_set_irq_enabled_with_callback(botao_a, GPIO_IRQ_EDGE_FALL, true,
&int_botao);
  gpio_set_irq_enabled_with_callback(botao_b, GPIO_IRQ_EDGE_FALL, true,
&int_botao);
  gpio_set_irq_enabled_with_callback(botao_c, GPIO_IRQ_EDGE_FALL, true,
&int_botao);
}

void gerarBloco(){          //gera um bloco de cor aleatória na linha zero e
coluna 2 da matriz
  x=0, y=2;
  int cor=(rand()%7)+1;
```

```
  led[x][y]=cor;
  piso=false;
  return;
}

void paraBaixo(){             //move o bloco atual para baixo
  if(led[x+1][y]==0){
    led[x+1][y]=led[x][y];
    led[x][y]=0;
    writeColor();
    x+=1;
    //printf("atual em: (%d, %d), valor de atual: %d\n", x, y, led[x][y]);
    piso=false;
  }else{
    piso=true;
  }
  return;
}

void paraDireita(){                  //move o bloco atual para direita
  if(y<=3 && led[x][y+1]==0){
    led[x][y+1]=led[x][y];
    led[x][y]=0;
    y+=1;
    writeColor();
  }
  return;
}

void paraEsquerda(){                   //move o bloco atual para esquerda
  if(y>=1 && led[x][y-1]==0){
    led[x][y-1]=led[x][y];
    led[x][y]=0;
    y-=1;
    writeColor();
  }
  return;
}

bool componente[10][10];
bool visitado[10][10];
int tamanho_comp=0;

void buscar_componente(int i, int j, int cor){      //encontra uma
componente de blocos e armazena resultados em componente e tamanho_comp
  componente[i][j]=true;
  visitado[i][j]=true;
  tamanho_comp++;

  int x1=i-1, y1=j;
```

```
  int x2=i, y2=j+1;
  int x3=i+1, y3=j;
  int x4=i, y4=j-1;

  if(x1>=1 && x1<=6 && y1>=0 && y1<=5) if(!visitado[x1][y1] && led[x1]
[y1]==cor) buscar_componente(x1, y1, cor);
  if(x2>=1 && x2<=6 && y2>=0 && y2<=5) if(!visitado[x2][y2] && led[x2]
[y2]==cor) buscar_componente(x2, y2, cor);
  if(x3>=1 && x3<=6 && y3>=0 && y3<=5) if(!visitado[x3][y3] && led[x3]
[y3]==cor) buscar_componente(x3, y3, cor);
  if(x4>=1 && x4<=6 && y4>=0 && y4<=5) if(!visitado[x4][y4] && led[x4]
[y4]==cor) buscar_componente(x4, y4, cor);
  return;
}

void reseta_componente(){          //reseta valores encontrados por
buscar_componente
  for(int i=0; i<7; i++){
    for(int j=0; j<7; j++){
      componente[i][j]=false;
      visitado[i][j]=false;
    }
  }
  tamanho_comp=0;
}

void apaga_componente(){             //apaga componente que foi encontrada
  for(int i=1; i<6; i++){
    for(int j=0; j<5; j++){
      if(componente[i][j]) led[i][j]=8;
    }
  }
  writeColor();
  sleep_ms(150);
  for(int i=1; i<6; i++){
    for(int j=0; j<5; j++){
      if(componente[i][j]){
        for(int k=i-1; k>0; k--) led[k+1][j]=led[k][j];
      }
    }
  }
  writeColor();
  sleep_ms(150);
}

void varredura(){                //percorre toda a matriz e faz buscas de
componentes nos elementos
  for(int i=1; i<6; i++){
    for(int j=0; j<5; j++){
      int cor=led[i][j];
```

```
      if(cor){
        reseta_componente();
        buscar_componente(i, j, cor);
        if(tamanho_comp>=3){
          pontos+=tamanho_comp;
          printf("PONTUACAO:   %d\n", pontos);
          apaga_componente();
          varredura();
        }
      }
    }
  }
}

bool tela_cheia(){      //retorna true se não há mais espaço vazio na matriz de LEDs
  if(x==0 && y==2) return true;
  if(led[1][1] && led[1][2] && led[1][3]) return true;
  return false;
}

void derrota(){               //animação da tela de derrota
  for(int i=5; i>=1; i--){
    for(int j=0; j<5; j++){
      if(led[i][j]){
        led[i][j]=8;
        writeColor();
        sleep_ms(150);
      }
    }
    for(int j=0; j<5; j++){
      led[i][j]=0;
      writeColor();
      sleep_ms(150);
    }
  }
}

int main() {

  //inicializações
  setup();
  srand(get_absolute_time()); //gera seed para rand()
  npInit(LED_PIN); // Inicializa a matriz de LEDs
  npClear();
  writeColor();
  gerarBloco();
  sleep_ms(1000);
  reseta_componente();
```

```
  while (true) {

    while(!piso && !intf){  //comportamento quando nada eh acionado
      paraBaixo();
      sleep_ms(speed);
      writeColor();
      sleep_ms(speed);
    }
    if(piso){               //se chegou no piso
      if(tela_cheia()){
        derrota();
        pontos=0;
      }
      varredura();
      gerarBloco();         //gera um novo bloco
      sleep_ms(speed);
    }
    else if(intf){          //interrupção gerada
      intf=false;
      if(flagBotao_a){      //pelo botão A
        flagBotao_a=false;
        paraEsquerda();
      }
      if(flagBotao_b){      //pelo botão B
        flagBotao_b=false;
        paraDireita();
      }
      if(flagBotao_c){      //pelo botão C
        flagBotao_c=false;
        paraBaixo();
      }

      sleep_ms(50);
      writeColor();
      sleep_ms(50);
    }
  }
}
```