

ANEXO III – CÓDIGO-FONTE

```

#include <Adafruit_GFX.h>
#include <Adafruit_PCD8544.h>
#include <math.h>
#include <LowPower.h>
#include <EEPROM.h> //EEPROM: [0]-snake.max_score, [1]-tetris.max_score,
[2]-breakout.max_score, [3]-spaceinvaders.max_score, [4]-backlight, [5]-difficulty
//Pins relativos ao display Nokia 5110
#define RST_PIN 12
#define CE_PIN 11
#define DC_PIN 10
#define DIN_PIN 9
#define CLK_PIN 8
#define BL_PIN 0
//Pins relativos aos botões
#define LEFT_BTN_PIN 6
#define RIGHT_BTN_PIN 4
#define GREEN_BTN_PIN 3
#define RED_BTN_PIN 5
//outras constates
#define CONTRAST 50
#define WIDTH 84 //comprimento do display (px)
#define HEIGHT 48 //altura do display (px)
#define SNAKE_GRID_X 21 //tamanho horizontal da grelha (snake)
#define SNAKE_GRID_Y (SNAKE_GRID_X*HEIGHT/WIDTH) //tamanho vertical da grelha (snake)
#define TETRIS_GRID_Y 16 //tamanho vertical da grelha (tetris)
#define TETRIS_GRID_X 10 //tamanho horizontal da grelha (tetris)
#define MAX_SNAKE_LENGTH SNAKE_GRID_X*SNAKE_GRID_Y/2 //comprimento máximo da snake (por
motivos de falta de memória não é possível aumentar este valor)
#define BREAKOUT_PLATFORM_LENGTH 6 //tamanho da plataforma (breakout)
#define REPETITION_DELAY 300 //atraso de repetição
#define REPETITION_SPEED_DELAY 50 //velocidade de repetição
#define BREAKOUT_GRID_X 21 //tamanho horizontal da grelha (breakout)
#define BREAKOUT_GRID_Y (BREAKOUT_GRID_X*HEIGHT/WIDTH) //tamanho vertical da grelha (breakout)
#define BREAKOUT_NEW_LEVEL (BREAKOUT_GRID_X*2) //numero de pontos necessários para avançar
de nível (breakout)
#define SPACEINVADERS_GRID_X 21 //tamanho horizontal da grelha (space invaders)
#define SPACEINVADERS_GRID_Y (SPACEINVADERS_GRID_X*HEIGHT/WIDTH) //tamanho vertical da
grelha (space invaders)

//variáveis snake
typedef struct
{
    int score; //pontuação
    int max_score; //pontuação máxima
    int v_x; //velocidade em x
    int v_y; //velocidade em y
    int dir; //direção
    int head_x; //coordenada x da cabeça
    int head_y; //coordenada y da cabeça
    int size; //tamanho da cobra
    int apple_x; //coordenada x da maçã
    int apple_y; //coordenada y da maçã
    char trail[MAX_SNAKE_LENGTH][2]; //coordenadas de todos os pontos do corpo da cobra
    bool over; //jogo terminado
    int game_delay; //intervalo de tempo que o jogo demora num ciclo
    bool paused; //jogo em pausa
}snake_info;

//variáveis tetris
typedef struct
{
    int next_piece; //codigo da próxima peça
    int current_piece; //codigo da peça atual
    int score; //pontuação
    int max_score; //pontuação máxima
    bool next_piece_drawing[4][2]; //desenho da próxima peça
    char current_piece_coords[4][2]; //coordenadas de cada bloco da peça atual
    bool grid[TETRIS_GRID_X][TETRIS_GRID_Y]; //grelha de jogo
    bool over; //jogo terminado
    int game_delay; //intervalo de tempo que o jogo demora num ciclo
    bool paused; //jogo em pausa

```

```

}tetris_info;

//variaveis breakout
typedef struct
{
    int platform_x; //coordenada x do bloco esquerdo da plataforma
    bool grid[BREAKOUT_GRID_X][BREAKOUT_GRID_Y]; //grelha de jogo
    float ball_x; //coordenada x da bola
    float ball_y; //coordenada y da bola
    float ball_v_x; //velocidade x da bola
    float ball_v_y; //velocidade y da bola
    bool over; //jogo terminado
    int score; //pontuação
    int max_score; //pontuação máxima
    int game_delay; //intervalo de tempo que o jogo demora num ciclo
    bool paused; //jogo em pausa
}breakout_info;

//variáveis spaceinvaders
typedef struct
{
    //aliens
    int count_alien; //número de aliens existentes
    bool aliens_move_left; // aliens movem-se para a esquerda
    bool aliens_move_right; //aliens movem-se para a direita
    bool aliens_move_down; //aliens movem-se para baixo
    //jogo
    bool over; //jogo terminado
    int cycle_counter; //contagem do número de ciclos
    bool paused; //jogo pausado
    int game_delay; //intervalo de tempo que o jogo demora num ciclo
    int score; //pontuação
    int max_score; //pontuação máxima
    char grid[SPACEINVADERS_GRID_X][SPACEINVADERS_GRID_Y]; //grelha de jogo
} spaceinvaders_info;

//inicialização do display
Adafruit_PCD8544 display = Adafruit_PCD8544(CLK_PIN, DIN_PIN, DC_PIN, CE_PIN, RST_PIN);

bool backlight = false; //luz de fundo
int current_game = -1; // -1-stand-by, 0-menus, 1-snake, 2-tetris, 3-breakout
int current_menu = 0; //0-menu principal, 1-jogos, 2-definições, 3-dificuldade
int selected_menu = 0;
bool show_upper = true; //quando os menus têm mais que 3 submenus, se show_upper = true,
mostram-se os 3 primeiros no ecrã
//estado anterior dos botões
bool left_btn_state_prev = LOW;
bool right_btn_state_prev = LOW;
bool green_btn_state_prev = LOW;
bool red_btn_state_prev = LOW;
//dificuldade
int difficulty = 1;
int new_difficulty = 1;
//quantidade de "tempo" que o utilizador pressiona as teclas
int green_pressed_time = 0;
int left_pressed_time = 0;
int right_pressed_time = 0;
//variáveis de iteração (definidas como variáveis globais para poupar memória)
int i = 0;
int j = 0;
int k = 0;

//inicialização dos jogos
snake_info snake = {0};
tetris_info tetris = {0};
spaceinvaders_info spaceinvaders = {0};
breakout_info breakout = {0};

void setup()
{

```

```
//configuração dos pinos
pinMode(LEFT_BTN_PIN, INPUT);
pinMode(RIGHT_BTN_PIN, INPUT);
pinMode(GREEN_BTN_PIN, INPUT);
pinMode(RED_BTN_PIN, INPUT);
pinMode(BL_PIN, OUTPUT);
srand(analogRead(1)); //semente para o algoritmo de geração de numeros aleatórios (o pino
1 não está conectado)
//configuração inicial do display
display.begin();
display.setContrast(CONTRAST);
display.clearDisplay();
display.display();
//reset de todos os jogos
resetSnake();
resetTetris();
resetBreakout();
resetSpaceInvaders();
//pontuações máximas e definições são lidas da memória persistente do arduino
for(i = 0; i < 5; i++)
{
    if(EEPROM.read(i) == 255)
        EEPROM.write(i, 0);
}
snake.max_score = EEPROM.read(0);
tetris.max_score = EEPROM.read(1);
breakout.max_score = EEPROM.read(2);
spaceinvaders.max_score = EEPROM.read(3);
backlight = (bool)EEPROM.read(4);
if(EEPROM.read(5) > 5 || EEPROM.read(5) == 0)
    EEPROM.write(i, 1);
difficulty = EEPROM.read(5);
new_difficulty = difficulty;
updateDelay();
}

void loop()
{
    //consola em stand-by
    if(current_game == -1)
    {
        display.clearDisplay();
        digitalWrite(BL_PIN, LOW);
        display.display();
        attachInterrupt(digitalPinToInterrupt(GREEN_BTN_PIN), wakeUp, HIGH); //a consola sai
        deste modo se pressionarmos o botão verde
        LowPower.powerDown(SLEEP_FOREVER, ADC_OFF, BOD_OFF); //modo powerdown (baixo consumo de
        energia)
        green_btn_state_prev = HIGH;
        current_game = 0;
        current_menu = 0;
        digitalWrite(BL_PIN, backlight);
        detachInterrupt(digitalPinToInterrupt(GREEN_BTN_PIN));
    }
    //menus
    else if(current_game == 0)
    {
        menu();
    }
    //snake
    else if(current_game == 1)
    {
        snake_game();
    }
    //tetris
    else if(current_game == 2)
    {
        tetris_game();
    }
    //breakout
    else if(current_game == 3)
```

```

    {
        breakout_game();
    }
    //space invaders
    else if(current_game == 4)
    {
        spaceInvaders_game();
    }
}

//funções relativas aos menus
-----
//função principal que lida com os menus
void menu()
{
    int num_of_menus = 0; //numero de sub-menus em cada menu
    //desenha no lcd
    drawMenu();
    //clique no botão direito
    if(checkRightBtnPress())
    {
        if(current_menu == 3)
            new_difficulty = new_difficulty + 1; //aumentar dificuldade (no menu dificuldade)
        else
            selected_menu = selected_menu + 1; //selecionar o menu abaixo
    }
    //clique no botão esquerdo
    if(checkLeftBtnPress())
    {
        if(current_menu == 3)
            new_difficulty = new_difficulty - 1; //diminuir dificuldade (no menu dificuldade)
        else
            selected_menu = selected_menu - 1; //selecionar o menu acima
    }
    //clique no botão verde
    if(checkGreenBtnPress())
    {
        if(current_menu == 0)
        {
            if(selected_menu == 0)
                current_menu = 1; //menu principal -> jogos
            else if(selected_menu == 1)
                current_menu = 2; //menu principal -> definições

            selected_menu = 0;
        }
        else if(current_menu == 1)
        {
            if(selected_menu == 0)
                current_game = 1; //jogos -> snake
            else if(selected_menu == 1)
                current_game = 2; //jogos -> tetris
            else if(selected_menu == 2)
                current_game = 3; //jogos -> breakout
            else if(selected_menu == 3)
                current_game = 4; //jogos -> spaceinvaders
        }
        else if(current_menu == 2)
        {
            if(selected_menu == 0)
                current_menu = 3; //definições -> dificuldade
            else if(selected_menu == 1)
            {
                backlight = !backlight; //definições -> luz de fundo on/off
                digitalWrite(BL_PIN, backlight);
                EEPROM.write(4, backlight);
            }
        }
        //menu dificuldade (guardar alterações e voltar atrás)
        else if(current_menu == 3)

```

```
{
    difficulty = new difficulty;
    EEPROM.write(5, difficulty);
    updateDelay();
    current_menu = 2;
}
}
//clique no botão vermelho
if(checkRedBtnPress())
{
    if(current_menu == 0)
        current_game = -1; //menu principal -> stand-by
    else if(current_menu == 1 || current_menu == 2)
    {
        current_menu = 0; //jogos/definições -> menu principal
        selected_menu = 0;
    }
    else if(current_menu == 3)
    {
        new_difficulty = difficulty;
        current_menu = 2; //dificuldade -> definições
        selected_menu = 0;
    }
}
//numero de sub-menus existentes em cada menu
if(current_menu == 0 || current_menu == 2)
{
    num_of_menus = 2;
}
else if(current_menu == 1)
{
    num_of_menus = 4;
}
//ajuste da variável selected_menu (0 a num_of_menus - 1)
if(selected_menu < 0)
{
    selected_menu = num_of_menus - 1;
}
if(selected_menu >= num_of_menus)
{
    selected_menu = 0;
}

//ajuste da variável new_difficulty (1 a 5)
if(new_difficulty < 1)
{
    new_difficulty = 5;
}
if(new_difficulty > 5)
{
    new_difficulty = 1;
}
}

//desenha no lcd o menu atual
void drawMenu()
{
    //menu principal
    if(current_menu == 0)
    {
        display.setTextSize(1);
        display.clearDisplay();
        display.setTextColor(BLACK, WHITE);
        display.setCursor(0, 1);
        display.print("Menu Principal");
        //barra separadora
        display.drawFastHLine(0, 10, WIDTH, BLACK);
        display.setCursor(0, 15);
        //menu selecionado fica com o fundo preto e letras brancas
        if(selected_menu == 0)
        {

```

```
    display.setTextColor(WHITE, BLACK);
}
else
{
    display.setTextColor(BLACK, WHITE);
}
display.print(">Jogos");
display.setCursor(0, 25);

if(selected_menu == 1)
{
    display.setTextColor(WHITE, BLACK);
}
else
{
    display.setTextColor(BLACK, WHITE);
}
display.print(">Definicoes");
}
//jogos
else if(current_menu == 1)
{
    if(selected_menu == 0)
        show_upper = true;
    if(selected_menu == 3)
        show_upper = false;

    display.setTextSize(1);
    display.clearDisplay();
    display.setTextColor(BLACK, WHITE);
    display.setCursor(25, 1);
    display.print("Jogos");
    display.drawFastHLine(0,10,WIDTH,BLACK);
    display.setCursor(0, 15);
    //3 primeiros menus
    if(show_upper)
    {
        if(selected_menu == 0)
        {
            display.setTextColor(WHITE, BLACK);
        }
        else
        {
            display.setTextColor(BLACK, WHITE);
        }
        display.print(">Snake");
        display.setCursor(0, 25);

        if(selected_menu == 1)
        {
            display.setTextColor(WHITE, BLACK);
        }
        else
        {
            display.setTextColor(BLACK, WHITE);
        }
        display.print(">Tetris");
        display.setCursor(0, 35);

        if(selected_menu == 2)
        {
            display.setTextColor(WHITE, BLACK);
        }
        else
        {
            display.setTextColor(BLACK, WHITE);
        }
        display.print(">Breakout");
    }
    //3 ultimos menus
    else
```

```
{
  if(selected_menu == 1)
  {
    display.setTextColor(WHITE, BLACK);
  }
  else
  {
    display.setTextColor(BLACK, WHITE);
  }
  display.print(">Tetris");
  display.setCursor(0, 25);

  if(selected_menu == 2)
  {
    display.setTextColor(WHITE, BLACK);
  }
  else
  {
    display.setTextColor(BLACK, WHITE);
  }
  display.print(">Breakout");
  display.setCursor(0, 35);

  if(selected_menu == 3)
  {
    display.setTextColor(WHITE, BLACK);
  }
  else
  {
    display.setTextColor(BLACK, WHITE);
  }
  display.print(">SpaceInvaders");
}

}
//definições
else if(current_menu == 2)
{
  display.setTextSize(1);
  display.clearDisplay();
  display.setTextColor(BLACK, WHITE);
  display.setCursor(15, 1);
  display.print("Definicoes");
  display.drawFastHLine(0,10,WIDTH,BLACK);
  display.setCursor(0, 15);

  if(selected_menu == 0)
  {
    display.setTextColor(WHITE, BLACK);
  }
  else
  {
    display.setTextColor(BLACK, WHITE);
  }
  display.print(">Dificuldade: ");
  display.setCursor(77, 15);
  display.print(difficulty);
  display.setCursor(0, 25);

  if(selected_menu == 1)
  {
    display.setTextColor(WHITE, BLACK);
  }
  else
  {
    display.setTextColor(BLACK, WHITE);
  }
  if(backlight)
    display.print(">Luz:      <ON>");
  else
    display.print(">Luz:      <OFF>");
}
```



```
    display.setCursor(0, 35);
}
//dificuldade
else if(current_menu == 3)
{
    display.setTextSize(1);
    display.clearDisplay();
    display.setTextColor(BLACK, WHITE);
    display.setCursor(10, 1);
    display.print("Dificuldade");
    display.drawFastHLine(0,10,WIDTH,BLACK);
    display.setTextSize(3);
    display.setCursor(15, 20);
    display.print("<");
    display.setCursor(55, 20);
    display.print(">");
    display.setCursor(35, 20);
    display.print(new_difficulty);
}
display.display();
}

//funções relativas ao jogo
snake-----
//reinicia a snake
void resetSnake()
{
    snake.v_x = snake.v_y = snake.dir = 0;
    //começa no centro da grelha
    snake.head_x = SNAKE_GRID_X/2;
    snake.head_y = SNAKE_GRID_Y/2;
    snake.size = 2;
    snake.trail[0][0] = SNAKE_GRID_X/2;
    snake.trail[0][1] = SNAKE_GRID_Y/2 + 1;
    snake.apple_x = -1;
    snake.apple_y = -1;
    snake.over = false;
    snake.score = 0;
    snake.paused = true;
}

//função principal que lida com o snake
void snake_game()
{
    bool left = false;
    bool right = false;
    //inicio do jogo - gerar maçã
    if(snake.apple_x == -1 && snake.apple_y == -1)
        generateApple();
    //o utilizador dispõe de tempo (game_delay) para fazer uma jogada
    for(k = 0; k < snake.game_delay; k++)
    {
        //botão esquerdo premido
        if(checkLeftBtnPress())
        {
            left = false;
            right = true;
            delay(REPETITION_SPEED_DELAY);
            k = k + REPETITION_SPEED_DELAY;
            snake.paused = false;
        }
        //botão direito premido
        if(checkRightBtnPress())
        {
            left = true;
            right = false;
            delay(REPETITION_SPEED_DELAY);
            k = k + REPETITION_SPEED_DELAY;
            snake.paused = false;
        }
        //botão vermelho premido
```

```
if(checkRedBtnPress())
{
    snake.paused = true;
    current_game = 0;
    //se o jogo tiver terminado, reinicia-lo
    if(snake.over)
    {
        resetSnake();
    }
}
delay(1);
}
//atualiza o jogo enquanto não tiver perdido ou na pausa
if(!snake.over && !snake.paused)
{
    updateSnake(left, right);
}
if(snake.size >= MAX_SNAKE_LENGTH)
    snake.over = true;
//desenha a grelha de jogo
drawSnake();
//mensagem de pausa
if(snake.paused)
    gamePausedText();
//mensagem de "game over"
if(snake.over)
{
    if(snake.size >= MAX_SNAKE_LENGTH)
        gameOverText(snake.score, snake.max_score, true); //jogo ganho
    else
        gameOverText(snake.score, snake.max_score, false); //jogo perdido
}
display.display();
}

//atualiza o snake
void updateSnake(bool left, bool right)
{
    if (left || right)
    {
        //a tecla esquerda faz a cobra mudar a sua direção num sentido e a direita no sentido
        //contrário
        if(left)
            (snake.dir)++;
        else if(right)
        {
            (snake.dir)--;
            if((snake.dir) < 0)
                (snake.dir) = 3;
        }
        (snake.dir) = (snake.dir)%4;
        //definição das velocidades em função da direção
        if((snake.dir) == 0)
            { snake.v_x = 1; snake.v_y = 0; }
        else if(snake.dir == 1)
            { snake.v_x = 0; snake.v_y = 1; }
        else if(snake.dir == 2)
            { snake.v_x = -1; snake.v_y = 0; }
        else if(snake.dir == 3)
            { snake.v_x = 0; snake.v_y = -1; }
    }
    //movimento da cobra
    if(snake.v_x != 0 || snake.v_y != 0)
    {
        //cada bloco da cauda toma o lugar do seguinte
        for(i = snake.size - 2; i > 0; i--)
        {
            snake.trail[i][0] = snake.trail[i-1][0];
            snake.trail[i][1] = snake.trail[i-1][1];
        }
        snake.trail[0][0] = snake.head_x;
```

```

    snake.trail[0][1] = snake.head_y;
    snake.head_x += snake.v_x;
    snake.head_y += snake.v_y;
}
//caso a cobra colida com a parede, atravessa-a e aparece do outro lado
if(snake.head_x >= SNAKE_GRID_X) snake.head_x = 0;
if(snake.head_x < 0) snake.head_x = SNAKE_GRID_X - 1;
if(snake.head_y >= SNAKE_GRID_Y) snake.head_y = 0;
if(snake.head_y < 0) snake.head_y = SNAKE_GRID_Y - 1;
//se a cobra colidir consigo própria, acaba o jogo
for(i = snake.size - 2; i > 0; i--)
{
    if(snake.head_x == snake.trail[i][0] && snake.head_y == snake.trail[i][1])
        snake.over = true;
}
//se colidir com a maçã, aumenta de tamanho e a pontuação aumenta
if(snake.head_x == snake.apple_x && snake.head_y == snake.apple_y)
{
    snake.size++;
    snake.trail[snake.size - 2][0] = snake.apple_x;
    snake.trail[snake.size - 2][1] = snake.apple_y;
    generateApple(); //nova maçã
    snake.score+=5;
    //atualiza a pontuação máxima
    if(snake.score > snake.max_score)
    {
        snake.max_score = snake.score;
        EEPROM.write(0, snake.max_score);
    }
}
}

//desenha a grelha da snake no lcd
void drawSnake()
{
    //tamanho dos blocos (px)
    int block_size = WIDTH/SNAKE_GRID_X;
    display.clearDisplay();
    //maçã
    display.fillRect(snake.apple_x * block_size, snake.apple_y *
    block_size, block_size, block_size, BLACK);
    //cabeça
    display.fillCircle(snake.head_x * block_size + block_size/2, snake.head_y * block_size +
    block_size/2, block_size/2, BLACK);
    //corpo
    for(i = snake.size - 2; i >= 0; i--)
    {
        display.drawCircle(snake.trail[i][0] * block_size + block_size/2, snake.trail[i][1] *
        block_size + block_size/2, block_size/2, BLACK);
    }
    //pontuação
    display.setTextSize(1);
    display.setTextColor(BLACK, WHITE);
    display.setCursor(0, 0);
    display.print(snake.score);
}

//gera uma maçã num sitio aleatório
void generateApple()
{
    snake.apple_x = -1;
    snake.apple_y = -1;
    //tenta colocar a maçã num sitio não ocupado (nem no canto superior esquerdo, devido à
    pontuação estar nesse sitio)
    do{
        snake.apple_x = rand()%SNAKE_GRID_X;
        snake.apple_y = rand()%SNAKE_GRID_Y;
    }while(isOccupied(snake.apple_x, snake.apple_y) || ((snake.apple_y == 0 || snake.apple_y
    == 1) && (snake.apple_x == 0 || snake.apple_x == 1 || snake.apple_x == 2)));
}

```

```
}

//verifica se o ponto (x,y) está ocupado com a snake
bool isOccupied(int x, int y)
{
    if(snake.head_x == x && snake.head_y == y)
        return true;
    for(i = 0; i < snake.size - 1; i++)
    {
        if(snake.trail[i][0] == x && snake.trail[i][1] == y)
            return true;
    }
    return false;
}

//funções relativas ao jogo
tetris-----

//reinicia o tetris
void resetTetris()
{
    //peça atual e seguinte indefinidas
    tetris.next_piece = -1;
    tetris.current_piece = -1;
    //limpa toda a grelha
    for(i = 0; i < TETRIS_GRID_X; i++)
    {
        for(j = 0; j < TETRIS_GRID_Y; j++)
        {
            tetris.grid[i][j] = false;
        }
    }
    //outras variáveis
    tetris.over = false;
    green_pressed_time = 0;
    left_pressed_time = 0;
    right_pressed_time = 0;
    tetris.score = 0;
    tetris.paused = true;
}

//função principal que lida com o tetris
void tetris_game()
{
    //inicio do jogo
    if(tetris.next_piece == -1)
    {
        //gera uma nova peça (e uma peça seguinte)
        spawnPiece();
    }
    //o utilizador dispõe de tempo (game_delay) para fazer uma ou mais jogadas antes que a
    peça desça
    for(k = 0; k < tetris.game_delay; k++)
    {
        //mede há quanto "tempo" o jogador está a premir a tecla verde
        if(digitalRead(GREEN_BTN_PIN) == HIGH && green_pressed_time < REPETITION_DELAY)
        {
            green_pressed_time++;
        }
        if(green_pressed_time >= REPETITION_DELAY)
        {
            tetris.game_delay = REPETITION_SPEED_DELAY;
        }
        //mede há quanto "tempo" o jogador está a premir a tecla esquerda
        if(digitalRead(LEFT_BTN_PIN) == HIGH)
        {
            if(left_pressed_time < REPETITION_DELAY)
                left_pressed_time++;
        }
        else
            left_pressed_time = 0;
    }
}
```

```
//mede há quanto "tempo" o jogador está a premir a tecla direita
if(digitalRead(RIGHT_BTN_PIN) == HIGH)
{
    if(right_pressed_time < REPETITION_DELAY)
        right_pressed_time++;
}
else
    right_pressed_time = 0;

//move a peça para a esquerda caso o jogador tenha clicado na tecla esquerda ou tenha
esta tecla pressionada há algum tempo
if((checkLeftBtnPress() || left_pressed_time >= REPETITION_DELAY) && !tetris.over)
{
    moveLeft();
    drawTetris();
    display.display();
    //adiciona um delay de repetição
    delay(REPETITION_SPEED_DELAY);
    k = k + REPETITION_SPEED_DELAY;
    tetris.paused = false;
}
//move a peça para a direita caso o jogador tenha clicado na tecla direita ou tenha esta
tecla pressionada há algum tempo
if((checkRightBtnPress() || right_pressed_time >= REPETITION_DELAY) && !tetris.over)
{
    moveRight();
    drawTetris();
    display.display();
    //adiciona um delay de repetição
    delay(REPETITION_SPEED_DELAY);
    k = k + REPETITION_SPEED_DELAY;
    tetris.paused = false;
}
//verifica se o jogador largou a tecla verde
if(checkGreenBtnUnpress() && !tetris.over)
{
    //se pressionou e largou a tecla de repende, a peça atual vira 90°
    if(green_pressed_time < REPETITION_DELAY && tetris.current_piece != 6 /*6 - quadrado
    (não virar)*/)
    {
        flip();
        delay(REPETITION_SPEED_DELAY);
        k = k + REPETITION_SPEED_DELAY;
    }

    updateDelay(); //se já tem a tecla pressionada há algum tempo, há que repôr a
    velocidade do jogo
    green_pressed_time = 0;
    drawTetris();
    display.display();
}

//se clicar na tecla vermelha
if(checkRedBtnPress())
{
    //pausa o jogo, volta ao menu
    current_game = 0;
    tetris.paused = true;
    if(tetris.over)
    {
        //se o jogo tiver acabado, reinicia-o
        resetTetris();
    }
}
delay(1);
} //fim do ciclo

//atualiza o jogo
if(!tetris.over && !tetris.paused)
    updateTetris();
```

```

//desenha a grelha no lcd
drawTetris();
//mensagem de pausa
if(tetris.paused)
    gamePausedText();
//mensagem de "game over", caso o jogo tenha terminado
if(tetris.over)
{
    gameOverText(tetris.score, tetris.max_score, false);
}
display.display();
}

//desenha a grelha do tetris no lcd
void drawTetris()
{
    int block_size = HEIGHT/TETRIS_GRID_Y; //tamanho de cada bloco (px)
    int margin = (WIDTH - TETRIS_GRID_X * block_size) / 2; //tamanho das margens (px)
    display.clearDisplay();
    //peças paradas
    for(i = 0; i < TETRIS_GRID_X; i++)
    {
        for(j = 0; j < TETRIS_GRID_Y; j++)
        {
            if(tetris.grid[i][j] == true)
                display.fillRect(margin + i * block_size, j * block_size, block_size, block_size, BLACK);
        }
    }
    //peça em movimento
    for(i = 0; i < 4; i++)
    {
        display.drawRect(margin + tetris.current_piece_coords[i][0] * block_size,
            tetris.current_piece_coords[i][1] * block_size, block_size, block_size, BLACK);
    }
    //margens
    display.drawFastVLine(margin - 1, 0, HEIGHT, BLACK);
    display.drawFastVLine(margin + TETRIS_GRID_X * block_size, 0, HEIGHT, BLACK);
    //peça seguinte
    display.drawRect(margin + TETRIS_GRID_X * block_size + (margin - 4*block_size)/2 -
        block_size, (TETRIS_GRID_Y - 2)*block_size/2 - block_size, 6*block_size, 4*block_size,
        BLACK);
    for(i = 0; i < 4; i++)
    {
        for(j = 0; j < 2; j++)
        {
            if(tetris.next_piece_drawing[i][j] == true)
                display.drawRect(margin + TETRIS_GRID_X * block_size + (margin - 4*block_size)/2 +
                    i*block_size, (TETRIS_GRID_Y - 2)*block_size/2 + j*block_size, block_size,
                    block_size, BLACK);
        }
    }
    //pontuação
    display.setTextSize(1);
    display.setTextColor(BLACK, WHITE);
    display.setCursor(0, 0);
    display.print(tetris.score);
}

//atualiza o tetris
void updateTetris()
{
    //verificar se é possível mover a peça para baixo
    for(i = 0; i < 4; i++)
    {
        if(tetris.current_piece_coords[i][1] == TETRIS_GRID_Y - 1 ||
            tetris.grid[tetris.current_piece_coords[i][0]][tetris.current_piece_coords[i][1] + 1] ==
            true)
        {
            //se não for, essa peça fica parada e é gerada uma nova
            for(i = 0; i < 4; i++)
            {

```

```
    if(tetris.current_piece_coords[i][1] >= 0)
        tetris.grid[tetris.current_piece_coords[i][0]][tetris.current_piece_coords[i][1]]
            = true;
    }
    spawnPiece(); //nova peça
    break;
}

//mover a peça para baixo
for(i = 0; i < 4; i++)
{
    tetris.current_piece_coords[i][1] = tetris.current_piece_coords[i][1] + 1;
}

//se os blocos parados chegarem ao topo da grelha, o jogo acaba
for(i = 0; i < TETRIS_GRID_X; i++)
{
    if(tetris.grid[i][0] == true)
        tetris.over = true;
}

//eliminar linhas completas
i = TETRIS_GRID_Y - 1;
while(i >= 0)
{
    //se existirem linhas completas
    if(checkFullLine(i))
    {
        eraseLine(i); //apagar linha
        tetris.score+=5; //incrementa pontuação
        //atualiza pontuação máxima
        if(tetris.score > tetris.max_score)
        {
            tetris.max_score = tetris.score;
            EEPROM.write(1, tetris.max_score);
        }
    }
    else
        i--;
}

//verifica se a linha "line" está completa
bool checkFullLine(int line)
{
    for(j = 0; j < TETRIS_GRID_X; j++)
    {
        if(tetris.grid[j][line] == false)
            return false;
    }
    return true;
}

//apaga a linha "line" e baixa todas as que está em cima
void eraseLine(int line)
{
    for(j = 0; j < TETRIS_GRID_X; j++)
    {
        for(k = line; k > 0; k--)
        {
            tetris.grid[j][k] = tetris.grid[j][k - 1];
        }
        tetris.grid[j][0] = false;
    }
}

//gera uma peça nova e uma peça seguinte
void spawnPiece()
{
    if(tetris.next_piece == -1)
```

```
tetris.next_piece = rand() % 7;
//a nova peça irá ser a seguinte
tetris.current_piece = tetris.next_piece;
//definição das coordenadas dos blocos da nova peça consoante o seu código
//peça em "L"
if(tetris.current_piece == 0)
{
    tetris.current_piece_coords[0][0] = TETRIS_GRID_X / 2;
    tetris.current_piece_coords[0][1] = -1;
    tetris.current_piece_coords[1][0] = TETRIS_GRID_X / 2 - 1;
    tetris.current_piece_coords[1][1] = -1;
    tetris.current_piece_coords[2][0] = TETRIS_GRID_X / 2 + 1;
    tetris.current_piece_coords[2][1] = -1;
    tetris.current_piece_coords[3][0] = TETRIS_GRID_X / 2 + 1;
    tetris.current_piece_coords[3][1] = -2;
}
//peça em "L" invertido
else if(tetris.current_piece == 1)
{
    tetris.current_piece_coords[0][0] = TETRIS_GRID_X / 2;
    tetris.current_piece_coords[0][1] = -1;
    tetris.current_piece_coords[1][0] = TETRIS_GRID_X / 2 - 1;
    tetris.current_piece_coords[1][1] = -1;
    tetris.current_piece_coords[2][0] = TETRIS_GRID_X / 2 + 1;
    tetris.current_piece_coords[2][1] = -1;
    tetris.current_piece_coords[3][0] = TETRIS_GRID_X / 2 - 1;
    tetris.current_piece_coords[3][1] = -2;
}
//peça em "S"
else if(tetris.current_piece == 2)
{
    tetris.current_piece_coords[0][0] = TETRIS_GRID_X / 2;
    tetris.current_piece_coords[0][1] = -1;
    tetris.current_piece_coords[1][0] = TETRIS_GRID_X / 2 - 1;
    tetris.current_piece_coords[1][1] = -1;
    tetris.current_piece_coords[2][0] = TETRIS_GRID_X / 2;
    tetris.current_piece_coords[2][1] = -2;
    tetris.current_piece_coords[3][0] = TETRIS_GRID_X / 2 + 1;
    tetris.current_piece_coords[3][1] = -2;
}
//peça em "Z"
else if(tetris.current_piece == 3)
{
    tetris.current_piece_coords[0][0] = TETRIS_GRID_X / 2;
    tetris.current_piece_coords[0][1] = -2;
    tetris.current_piece_coords[1][0] = TETRIS_GRID_X / 2 - 1;
    tetris.current_piece_coords[1][1] = -2;
    tetris.current_piece_coords[2][0] = TETRIS_GRID_X / 2;
    tetris.current_piece_coords[2][1] = -1;
    tetris.current_piece_coords[3][0] = TETRIS_GRID_X / 2 + 1;
    tetris.current_piece_coords[3][1] = -1;
}
//peça em "T"
else if(tetris.current_piece == 4)
{
    tetris.current_piece_coords[0][0] = TETRIS_GRID_X / 2;
    tetris.current_piece_coords[0][1] = -1;
    tetris.current_piece_coords[1][0] = TETRIS_GRID_X / 2 - 1;
    tetris.current_piece_coords[1][1] = -1;
    tetris.current_piece_coords[2][0] = TETRIS_GRID_X / 2 + 1;
    tetris.current_piece_coords[2][1] = -1;
    tetris.current_piece_coords[3][0] = TETRIS_GRID_X / 2;
    tetris.current_piece_coords[3][1] = -2;
}
//peça em "I"
else if(tetris.current_piece == 5)
{
    tetris.current_piece_coords[0][0] = TETRIS_GRID_X / 2;
    tetris.current_piece_coords[0][1] = -1;
    tetris.current_piece_coords[1][0] = TETRIS_GRID_X / 2 - 1;
    tetris.current_piece_coords[1][1] = -1;
```



```
tetris.current_piece_coords[2][0] = TETRIS_GRID_X / 2 - 2;
tetris.current_piece_coords[2][1] = -1;
tetris.current_piece_coords[3][0] = TETRIS_GRID_X / 2 + 1;
tetris.current_piece_coords[3][1] = -1;
}
//peça em quadrado
else if(tetris.current_piece == 6)
{
    tetris.current_piece_coords[0][0] = TETRIS_GRID_X / 2;
    tetris.current_piece_coords[0][1] = -1;
    tetris.current_piece_coords[1][0] = TETRIS_GRID_X / 2 - 1;
    tetris.current_piece_coords[1][1] = -1;
    tetris.current_piece_coords[2][0] = TETRIS_GRID_X / 2;
    tetris.current_piece_coords[2][1] = -2;
    tetris.current_piece_coords[3][0] = TETRIS_GRID_X / 2 - 1;
    tetris.current_piece_coords[3][1] = -2;
}
//gerar peça seguinte aleatória
tetris.next_piece = rand() % 7;

for(i = 0; i < 4; i++)
{
    tetris.next_piece_drawing[i][0] = false;
    tetris.next_piece_drawing[i][1] = false;
}
//desenha a peça seguinte conforme o seu código
//peça em "L"
if(tetris.next_piece == 0)
{
    tetris.next_piece_drawing[0][1] = true;
    tetris.next_piece_drawing[1][1] = true;
    tetris.next_piece_drawing[2][1] = true;
    tetris.next_piece_drawing[2][0] = true;
}
//peça em "L" invertido
else if(tetris.next_piece == 1)
{
    tetris.next_piece_drawing[0][1] = true;
    tetris.next_piece_drawing[1][1] = true;
    tetris.next_piece_drawing[2][1] = true;
    tetris.next_piece_drawing[0][0] = true;
}
//peça em "S"
else if(tetris.next_piece == 2)
{
    tetris.next_piece_drawing[0][1] = true;
    tetris.next_piece_drawing[1][1] = true;
    tetris.next_piece_drawing[1][0] = true;
    tetris.next_piece_drawing[2][0] = true;
}
//peça em "Z"
else if(tetris.next_piece == 3)
{
    tetris.next_piece_drawing[0][0] = true;
    tetris.next_piece_drawing[1][0] = true;
    tetris.next_piece_drawing[1][1] = true;
    tetris.next_piece_drawing[2][1] = true;
}
//peça em "T"
else if(tetris.next_piece == 4)
{
    tetris.next_piece_drawing[0][1] = true;
    tetris.next_piece_drawing[1][1] = true;
    tetris.next_piece_drawing[1][0] = true;
    tetris.next_piece_drawing[2][1] = true;
}
//peça em "|"
else if(tetris.next_piece == 5)
{
    tetris.next_piece_drawing[0][0] = true;
    tetris.next_piece_drawing[1][0] = true;
```

```

    tetris.next_piece_drawing[2][0] = true;
    tetris.next_piece_drawing[3][0] = true;
}
//peça em quadrado
else if(tetris.next_piece == 6)
{
    tetris.next_piece_drawing[0][0] = true;
    tetris.next_piece_drawing[0][1] = true;
    tetris.next_piece_drawing[1][0] = true;
    tetris.next_piece_drawing[1][1] = true;
}
}

//move a peça para a esquerda se possível
void moveLeft()
{
    //verificar se é possível mover
    for(i = 0; i < 4; i++)
    {
        if(tetris.current_piece_coords[i][0] == 0 ||
        tetris.grid[tetris.current_piece_coords[i][0] - 1][tetris.current_piece_coords[i][1]] ==
        true)
            return;
    }
    //mover
    for(i = 0; i < 4; i++)
    {
        tetris.current_piece_coords[i][0] = tetris.current_piece_coords[i][0] - 1;
    }
}

//move a peça para a direita se possível
void moveRight()
{
    //verificar se é possível mover
    for(i = 0; i < 4; i++)
    {
        if(tetris.current_piece_coords[i][0] == TETRIS_GRID_X - 1 ||
        tetris.grid[tetris.current_piece_coords[i][0] + 1][tetris.current_piece_coords[i][1]] ==
        true)
            return;
    }
    //mover
    for(i = 0; i < 4; i++)
    {
        tetris.current_piece_coords[i][0] = tetris.current_piece_coords[i][0] + 1;
    }
}

//gira a peça 90° no sentido horário
void flip()
{
    int new_x;
    int new_y;
    //verificar se é possível girar
    for(i = 1; i < 4; i++)
    {
        new_x = tetris.current_piece_coords[0][0] + (tetris.current_piece_coords[0][1] -
        tetris.current_piece_coords[i][1]);
        new_y = tetris.current_piece_coords[0][1] + (tetris.current_piece_coords[0][0] -
        tetris.current_piece_coords[i][0]);
        if(new_x < 0 || new_x >= TETRIS_GRID_X || new_y >= TETRIS_GRID_Y || (new_y <= 0 &&
        tetris.grid[new_x][new_y] == true))
            return;
    }
    //girar (a peça do meio mantém sempre o seu lugar numa viragem)
    for(i = 1; i < 4; i++)
    {
        new_x = tetris.current_piece_coords[0][0] + (tetris.current_piece_coords[0][1] -
        tetris.current_piece_coords[i][1]);
        new_y = tetris.current_piece_coords[0][1] - (tetris.current_piece_coords[0][0] -

```

```
tetris.current_piece_coords[i][0]);
tetris.current_piece_coords[i][0] = new_x;
tetris.current_piece_coords[i][1] = new_y;
}
}

//funções relativas ao jogo
breakout-----
//reinicia o breakout
void resetBreakout()
{
    //plataforma no meio da grelha
    breakout.platform_x = (BREAKOUT_GRID_X - BREAKOUT_PLATFORM_LENGTH)/2 ;

    //conjunto de peças (até meio da grelha)
    for(i = 0; i < BREAKOUT_GRID_X; i++)
    {
        for(j = 0; j < BREAKOUT_GRID_Y; j++)
        {
            if(j < BREAKOUT_GRID_Y/2)
                breakout.grid[i][j] = true;
            else
                breakout.grid[i][j] = false;
        }
    }
    //restantes variáveis
    breakout.ball_x = BREAKOUT_GRID_X/2;
    breakout.ball_y = BREAKOUT_GRID_Y*3/4;
    breakout.ball_v_x = 0;
    breakout.ball_v_y = 1;
    breakout.over = false;
    breakout.score = 0;
    breakout.paused = true;
}

//função principal que lida com o breakout
void breakout_game()
{
    bool left = false;
    bool right = false;
    //o utilizador dispõe de tempo (game_delay) para fazer uma jogada
    for(k = 0; k < breakout.game_delay; k++)
    {
        //clicar na tecla esquerda
        if(digitalRead(LEFT_BTN_PIN) == HIGH)
        {
            left = true;
            right = false;
            breakout.paused = false;
        }
        //clicar na tecla direita
        if(digitalRead(RIGHT_BTN_PIN) == HIGH)
        {
            left = false;
            right = true;
            breakout.paused = false;
        }
        //clicar na tecla vermelha
        if(checkRedBtnPress())
        {
            //pausa o jogo, volta ao menu
            current_game = 0;
            if(breakout.over)
            {
                //se o jogo tiver acabado, reinicia-o
                resetBreakout();
            }
            breakout.paused = true;
        }
        delay(1);
    }
}
```

```

//Se o jogo estiver a decorrer, atualiza-o
if(!breakout.over && !breakout.paused)
{
    updateBreakout(left, right);
}
//desenha o jogo no lcd
drawBreakout();
//mensagem de pausa
if(breakout.paused)
    gamePausedText();
if(checkBreakoutWin())
    breakout.over = true;
//mostra uma mensagem de "game over" se tiver terminado
if(breakout.over)
{
    if(checkBreakoutWin())
        gameOverText(breakout.score, breakout.max_score, true);
    else
        gameOverText(breakout.score, breakout.max_score, false);
}
display.display();
}

//verifica se o jogo foi ganho (já não existem peças para destruir)
bool checkBreakoutWin()
{
    for(i = 0; i < BREAKOUT_GRID_X; i++)
    {
        for(j = 0; j < BREAKOUT_GRID_Y; j++)
        {
            if(breakout.grid[i][j] == true)
                return false;
        }
    }
    return true;
}

//atualiza o jogo
void updateBreakout(bool left, bool right)
{
    //move a plataforma conforme o input do utilizador
    if(left && breakout.platform_x > 0)
    {
        breakout.platform_x--;
    }
    if(right && breakout.platform_x + BREAKOUT_PLATFORM_LENGTH < BREAKOUT_GRID_X)
    {
        breakout.platform_x++;
    }

    //move a bola, tendo em conta a sua velocidade
    breakout.ball_x = (breakout.ball_x + breakout.ball_v_x);
    breakout.ball_y = (breakout.ball_y + breakout.ball_v_y);

    //verifica colisões da bola com:
    //paredes esquerda e direita
    if(((int)(roundf(breakout.ball_x)) <= 0 && breakout.ball_v_x < 0) ||
        ((int)(roundf(breakout.ball_x)) >= BREAKOUT_GRID_X - 1 && breakout.ball_v_x > 0))
        breakout.ball_v_x = -breakout.ball_v_x; //altera a direção
    //parede de cima
    if(((int)(roundf(breakout.ball_y)) <= 0 && breakout.ball_v_y < 0)
        breakout.ball_v_y = -breakout.ball_v_y; //altera a direção

    //peças à direita (se se estiver a mover para a direita)
    if(((int)(roundf(breakout.ball_x)) + 1 <= BREAKOUT_GRID_X - 1 &&
        breakout.grid[(int)(roundf(breakout.ball_x)) + 1][(int)(roundf(breakout.ball_y))] == true
        && breakout.ball_v_x > 0)
    {
        breakout.grid[(int)(roundf(breakout.ball_x)) + 1][(int)(roundf(breakout.ball_y))] =

```

```

    false; //elimina a peça
    breakoutHandleScore(); //lida com a pontuação
    breakout.ball_v_x = -breakout.ball_v_x; //altera a direção
}
//peças à esquerda (se se estiver a mover para a esquerda)
if((int)(roundf(breakout.ball_x)) - 1 >= 0 && breakout.grid[(int)(roundf(breakout.ball_x))
- 1][(int)(roundf(breakout.ball_y))] == true && breakout.ball_v_x < 0)
{
    breakout.grid[(int)(roundf(breakout.ball_x)) - 1][(int)(roundf(breakout.ball_y))] =
    false; //elimina a peça
    breakoutHandleScore(); //lida com a pontuação
    breakout.ball_v_x = -breakout.ball_v_x; //altera a direção
}
//peças em baixo (se se estiver a mover para baixo)
if((int)(roundf(breakout.ball_y)) + 1 <= BREAKOUT_GRID_Y - 1 &&
breakout.grid[(int)(roundf(breakout.ball_x))][(int)(roundf(breakout.ball_y)) + 1] == true
&& breakout.ball_v_y > 0)
{
    breakout.grid[(int)(roundf(breakout.ball_x))][(int)(roundf(breakout.ball_y)) + 1] =
    false; //elimina a peça
    breakoutHandleScore(); //lida com a pontuação
    breakout.ball_v_y = -breakout.ball_v_y; //altera a direção
}
//peças em cima (se se estiver a mover para cima)
if((int)(roundf(breakout.ball_y)) - 1 >= 0 &&
breakout.grid[(int)(roundf(breakout.ball_x))][(int)(roundf(breakout.ball_y)) - 1] == true
&& breakout.ball_v_y < 0)
{
    breakout.grid[(int)(roundf(breakout.ball_x))][(int)(roundf(breakout.ball_y)) - 1] =
    false; //elimina a peça
    breakoutHandleScore(); //lida com a pontuação
    breakout.ball_v_y = -breakout.ball_v_y; //altera a direção
}
//peças no próprio sitio em que a bola se encontra
if(breakout.grid[(int)(roundf(breakout.ball_x))][(int)(roundf(breakout.ball_y))] == true)
{
    breakout.grid[(int)(roundf(breakout.ball_x))][(int)(roundf(breakout.ball_y))] = false;
    //elimina a peça
    breakoutHandleScore(); //lida com a pontuação
}

//se alguma peça chegar à penultima linha da grelha, acaba o jogo
for(i = 0; i < BREAKOUT_GRID_X; i++)
{
    if(breakout.grid[i][BREAKOUT_GRID_Y - 2] == true)
    {
        breakout.over = true;
    }
}
//se a bola chegar à punultima linha da grelha, verifica-se se caiu em cima da plataforma
if((int)(roundf(breakout.ball_y)) == BREAKOUT_GRID_Y - 2)
{
    platformHit((int)(roundf(breakout.ball_x)));
}
//se a bola chegar à ultima linha da plataforma, acaba o jogo
if((int)(roundf(breakout.ball_y)) == BREAKOUT_GRID_Y - 1)
{
    breakout.over = true;
}
}

//lida com a pontuação após a colisão da bola com uma peça
void breakoutHandleScore()
{
    breakout.score++; //aumenta a pontuação
    //atualiza a pontuação máxima e a memória persistente relativa
    if(breakout.score > breakout.max_score)
    {
        breakout.max_score = breakout.score;
        EEPROM.write(2, breakout.max_score);
    }
}

```

```

//se o jogador tiver chegado a uma pontuação multipla de BREAKOUT_NEW_LEVEL, é criada uma
nova linha de peças
if(breakout.score%BREAKOUT_NEW_LEVEL == 0)
    dropOneLine();
}

//cria uma nova linha de peças na parte superior da grelha
void dropOneLine()
{
    //baixa todas as peças uma unidade
    for(i = 0; i < BREAKOUT_GRID_X; i++)
    {
        for(j = BREAKOUT_GRID_Y - 1 ; j > 0; j--)
        {
            breakout.grid[i][j] = breakout.grid[i][j - 1];
        }
        breakout.grid[i][0] = true; //novas peças
    }
    //baixa também a bola
    breakout.ball_y++;
}

//verifica se a bola atingiu a plataforma
void platformHit(int x)
{
    if(x >= breakout.platform_x && x <= breakout.platform_x + BREAKOUT_PLATFORM_LENGTH - 1)
    {
        //caso tenha atingido, é-lhe impingida uma nova velocidade, dependendo do sitio onde a
        atingiu
        breakout.ball_v_x = (x - breakout.platform_x)*sqrt(2)/(BREAKOUT_PLATFORM_LENGTH - 1)
        - sqrt(2) /2;
        breakout.ball_v_y = -sqrt(1 - breakout.ball_v_x * breakout.ball_v_x); //velocidade
        da bola (v_x^2 + v_y^2) = 1
    }
}

//desenha no lcd a grelha do breakout
void drawBreakout()
{
    display.clearDisplay();
    int block_size = HEIGHT/BREAKOUT_GRID_Y; //tamanho de cada peça (px)
    //blocos
    for(i = 0; i < BREAKOUT_GRID_X; i++)
    {
        for(j = 0; j < BREAKOUT_GRID_Y; j++)
        {
            if(breakout.grid[i][j] == true)
                display.fillRect(i*block_size, j*block_size, block_size, block_size, BLACK);
        }
    }
    //plataforma
    display.fillRect(breakout.platform_x * block_size, (BREAKOUT_GRID_Y-1)*block_size,
    BREAKOUT_PLATFORM_LENGTH*block_size, block_size, BLACK);
    //bola
    display.drawRect(breakout.ball_x * block_size, breakout.ball_y * block_size, block_size,
    block_size, BLACK);
    //pontuação
    display.setTextSize(1);
    display.setTextColor(BLACK, WHITE);
    display.setCursor(0, 0);
    display.print(breakout.score);
}

//funções relativas ao jogo spaceinvaders
-----
/*
-----
SPACE INVADERS
-----
Códigos da matriz bidimensional:

```

```
1 simboliza a nave
2 simboliza o alien
3 simboliza uma bala da nave
4 simboliza uma bala de alien

*/
//função principal que lida com o jogo space invaders
void spaceInvaders_game()
{

    bool right = false; //nave move-se para a direita
    bool left = false; //nave move-se para a esquerda
    bool shoot_order = false; //nave dispara

    for(k = 0; k < spaceinvaders.game_delay; k++)
    {
        //jogador clica/pressiona o botão esquerdo
        if(digitalRead(LEFT_BTN_PIN))
        {
            right = false;
            left = true;
            spaceinvaders.paused = false;
        }
        //jogador clica/pressiona o botão direito
        if(digitalRead(RIGHT_BTN_PIN))
        {
            right = true;
            left = false;
            spaceinvaders.paused = false;
        }
        //jogador clica/pressiona o botão esquerdo
        if(checkGreenBtnPress()) // se o utilizador clicar no botão direito a nave dispara
        {
            shoot_order = true;
        }
        //jogador clica no botão vermelho
        if(checkRedBtnPress())
        {
            current_game = 0; //pausa o jogo, volta ao menu
            if(spaceinvaders.over)
            {
                //se o jogo tiver acabado, reinicia-o
                resetSpaceInvaders();
            }
            spaceinvaders.paused = true;
        }
        delay(1);
    }

    spaceinvaders.cicle_counter++; //aumenta o número de ciclos

    //atualiza o jogo (se não tiver acabado nem pausado)
    if(!spaceinvaders.over && !spaceinvaders.paused)
    {
        checkMoveAliens(); //altera, se necessário, a direção dos aliens
        checkCollisions(); //verifica as colisões

        if(spaceinvaders.cicle_counter%(-2*difficulty+14) == 0) //os aliens movem-se mais lentamente que as outras entidades, mas a sua velocidade aumenta com a dificuldade
        {
            moveAliens();
        }
        moveShip(left, right); //mover a nave
        checkCollisions(); //verificar novamente as colisões
        moveBullets(); //mover as balas

        if(spaceinvaders.cicle_counter%(-2*difficulty+14) == (-2*difficulty+14)/2 && spaceinvaders.count_alien > 0) //os aliens disparam balas à mesma velocidade a que se movem, mas enquanto estão parados
```

```

        createAlienBullet();
    if(shoot_order)
        createShipBullet();

}

drawSpace(); //desenhar o jogo
if(spaceinvaders.count.aliens == 0)
    spaceinvaders.over = true;
//mensagem de pausa
if(spaceinvaders.paused)
    gamePausedText();
//mensagem de "game over"
if(spaceinvaders.over)
{
    if(spaceinvaders.count.aliens == 0)
        gameOverText(spaceinvaders.score, spaceinvaders.max_score, true); //jogo ganho
    else
        gameOverText(spaceinvaders.score, spaceinvaders.max_score, false); //jogo perdido
}

display.display();
}

//reinicia o jogo
void resetSpaceInvaders()
{
    spaceinvaders.score = 0;
    spaceinvaders.cicle_counter = 0;
    spaceinvaders.count.aliens = 0;
    spaceinvaders.aliens_move_right = true;
    spaceinvaders.aliens_move_left = false;
    spaceinvaders.aliens_move_down = false;
    spaceinvaders.over = false;
    spaceinvaders.paused = true;
    spaceinvaders.game_delay = 150;
    loadSpace();
}

//guarda na grelha de jogo a posição inicial das entidades do jogo
void loadSpace()
{
    //pôr tudo a zeros no início
    for(i = 0; i < SPACEINVADERS_GRID_X; i++)
    {
        for(j = 0; j < SPACEINVADERS_GRID_Y; j++)
        {
            spaceinvaders.grid[i][j] = 0;
        }
    }

    for (i = 2; i < SPACEINVADERS_GRID_X - 2 ; i = i+2)
    {
        for (j = 0; j < SPACEINVADERS_GRID_Y/2; j = j+2)
        {
            spaceinvaders.grid[i][j] = 2; // definir a posição dos aliens (de dois
            dem dois blocos, vertical e horizontalmente, até meio da grelha)
            spaceinvaders.count.aliens++; // aumentar a contagem do número de aliens
        }
    }

    // definir a posição inicial da nave , representada pelo número 1 e por um triângulo
    spaceinvaders.grid[SPACEINVADERS_GRID_X/2][SPACEINVADERS_GRID_Y - 1] = 1;
    spaceinvaders.grid[SPACEINVADERS_GRID_X/2+1][SPACEINVADERS_GRID_Y - 1] = 1;
    spaceinvaders.grid[SPACEINVADERS_GRID_X/2-1][SPACEINVADERS_GRID_Y - 1] = 1;
    spaceinvaders.grid[SPACEINVADERS_GRID_X/2][SPACEINVADERS_GRID_Y - 2] = 1;
}

//função que cria uma bala da nave
void createShipBullet()
{

```



```

for(i = 1; i < SPACEINVADERS_GRID_X - 1; i++)
{
    if(spaceinvaders.grid[i-1][SPACEINVADERS_GRID_Y - 1] == 1 &&
spaceinvaders.grid[i+1][SPACEINVADERS_GRID_Y - 1] == 1) // identificar a posição do
centro da nave
    {
        if(spaceinvaders.grid[i][SPACEINVADERS_GRID_Y - 3] == 0)
            spaceinvaders.grid[i][SPACEINVADERS_GRID_Y - 3] = 3; // criar uma bala na ponta do
triângulo da nave
        else
            spaceinvaders.grid[i][SPACEINVADERS_GRID_Y - 3] = 0; //se houver uma bala inimiga
naquele sítio destruir a bala inimiga (e não se cria bala)
    }
}

//função que cria uma bala de alien
void createAlienBullet()
{
    int bullets_created = 0;
    do{

        i = rand()%SPACEINVADERS_GRID_X; //escolhe uma coluna aleatoria

        for (j = SPACEINVADERS_GRID_Y-1; j >= 0; j--) //verificar se existe algum alien na
coluna escolhida
        {
            if (spaceinvaders.grid[i][j] == 2)
            {
                if(spaceinvaders.grid[i][j+1] == 0)
                {
                    spaceinvaders.grid[i][j+1] = 4; //criar uma bala nova
                }
                else if(spaceinvaders.grid[i][j+1] == 3)
                {
                    spaceinvaders.grid[i][j+1] = 0; //se houver uma bala inimiga na casa a seguir
ambas as balas são destruídas
                }
                bullets_created++;
                break;
            }
        }
        }while(bullets_created < ceil(difficulty/2)); //o número de balas criadas não é
necessariamente ceil(difficulty/2), mas está certamente entre 1 e ceil(difficulty/2)
    }

//verifica as colisões entre as diferentes entidades
void checkCollisions()
{
    for(i = 0; i < SPACEINVADERS_GRID_X; i++)
    {
        //colisao das balas com as paredes
        if(spaceinvaders.grid[i][0] == 3)
            spaceinvaders.grid[i][0] = 0;
        if(spaceinvaders.grid[i][SPACEINVADERS_GRID_Y - 1] == 4)
            spaceinvaders.grid[i][SPACEINVADERS_GRID_Y - 1] = 0;

        if(spaceinvaders.aliens_move_down && spaceinvaders.grid[i][SPACEINVADERS_GRID_Y - 4] == 2)
            spaceinvaders.over = true;

        for(j = 0; j < SPACEINVADERS_GRID_Y - 1; j++)
        {
            //colisao de duas balas
            if(spaceinvaders.grid[i][j] == 4 && spaceinvaders.grid[i][j+1] == 3)
            {
                spaceinvaders.grid[i][j] = 0;
                spaceinvaders.grid[i][j+1] = 0;
            }
            //colisão de uma bala da nave com um alien

```

```

    if(spaceinvaders.grid[i][j] == 2 && spaceinvaders.grid[i][j+1] == 3)
    {
        spaceinvaders.grid[i][j] = 0;
        spaceinvaders.grid[i][j+1] = 0;
        spaceinvaders.count_aliens--;
        spaceinvaders.score += 5; //aumenta a pontuação
        //atualiza a pontuação máxima
        if(spaceinvaders.score > spaceinvaders.max_score)
        {
            spaceinvaders.max_score = spaceinvaders.score;
            EEPROM.write(3, spaceinvaders.max_score);
        }
    }
    //colisão de uma bala de alien com a nave
    if(spaceinvaders.grid[i][j] == 4 && spaceinvaders.grid[i][j+1] == 1)
    {
        spaceinvaders.grid[i][j] = 0;
        spaceinvaders.over = true;
    }
}
}
}

//movimenta as balas de aliens e da nave
void moveBullets()
{
    for(i = 0; i < SPACEINVADERS_GRID_X; i++)
    {
        for(j = 0; j < SPACEINVADERS_GRID_Y; j++)
        {
            if(spaceinvaders.grid[i][j] == 3) //bala da nave
            {
                spaceinvaders.grid[i][j-1] = 3;
                spaceinvaders.grid[i][j] = 0;
            }
        }
    }
    for(i = 0; i < SPACEINVADERS_GRID_X; i++)
    {
        for(j = SPACEINVADERS_GRID_Y - 1; j >= 0; j--)
        {
            if(spaceinvaders.grid[i][j] == 4) //bala de alien
            {
                spaceinvaders.grid[i][j+1] = 4;
                spaceinvaders.grid[i][j] = 0;
            }
        }
    }
}

//movimenta os aliens
void moveAliens()
{
    //movimenta os aliens para baixo uma unidade
    if(spaceinvaders.aliens_move_down)
    {
        for(i = 0; i < SPACEINVADERS_GRID_X; i++)
        {
            for(j = SPACEINVADERS_GRID_Y - 1; j >= 0; j--)
            {
                if(spaceinvaders.grid[i][j] == 2)
                {
                    spaceinvaders.grid[i][j + 1] = 2;
                    spaceinvaders.grid[i][j] = 0;
                }
            }
        }
        spaceinvaders.aliens_move_down = false; //apenas se movem uma unidade para baixo de cada

```

```

    vez que chegam a um dos limites da grelha
}
//movimenta os aliens para a direita uma unidade
else if(spaceinvaders.aliens_move_right)
{
    for(i = SPACEINVADERS_GRID_X - 1; i >= 0; i--)
    {
        for(j = 0; j < SPACEINVADERS_GRID_Y; j++)
        {
            if(spaceinvaders.grid[i][j] == 2)
            {
                //se esse espaço estiver ocupado com uma bala da nave, destrói o alien e a nave
                if(spaceinvaders.grid[i+1][j] == 3)
                {
                    spaceinvaders.grid[i+1][j] = 0;
                    spaceinvaders.grid[i][j] = 0;
                    spaceinvaders.count.aliens--;
                    spaceinvaders.score += 5; //aumenta a pontuação
                    //atualiza a pontuação máxima
                    if(spaceinvaders.score > spaceinvaders.max_score)
                    {
                        spaceinvaders.max_score = spaceinvaders.score;
                        EEPROM.write(3, spaceinvaders.max_score);
                    }
                }
            }
            else
            {
                spaceinvaders.grid[i+1][j] = 2;
                spaceinvaders.grid[i][j] = 0;
            }
        }
    }
}
//movimenta os aliens para a esquerda uma unidade
else if(spaceinvaders.aliens_move_left)
{
    for(i = 0; i < SPACEINVADERS_GRID_X; i++)
    {
        for(j = 0; j < SPACEINVADERS_GRID_Y; j++)
        {
            if(spaceinvaders.grid[i][j] == 2)
            {
                //se esse espaço estiver ocupado com uma bala da nave, destrói o alien e a nave
                if(spaceinvaders.grid[i-1][j] == 3)
                {
                    spaceinvaders.grid[i-1][j] = 0;
                    spaceinvaders.grid[i][j] = 0;
                    spaceinvaders.count.aliens--;
                    spaceinvaders.score += 5; //aumenta a pontuação
                    //atualiza a pontuação máxima
                    if(spaceinvaders.score > spaceinvaders.max_score)
                    {
                        spaceinvaders.max_score = spaceinvaders.score;
                        EEPROM.write(3, spaceinvaders.max_score);
                    }
                }
            }
            else
            {
                spaceinvaders.grid[i-1][j] = 2;
                spaceinvaders.grid[i][j] = 0;
            }
        }
    }
}
}

//função utilizada para alterar o sentido do movimento dos aliens quando chegarem ao limite
lateral esquerdo ao direito

```

```

void checkMoveAliens()
{
    int y;

    if(spaceinvaders.aliens_move_left)
    {
        for (y = 0; y < SPACEINVADERS_GRID_Y; y++)
        {
            //se existirem aliens no limite esquerdo
            if (spaceinvaders.grid[0][y] == 2)
            {
                //alteram o seu movimento, primeiro para baixo e depois para a direita
                spaceinvaders.aliens_move_right = true;
                spaceinvaders.aliens_move_left = false;
                spaceinvaders.aliens_move_down = true;
                break;
            }
        }
    }
    else if(spaceinvaders.aliens_move_right)
    {
        for (y = 0; y < SPACEINVADERS_GRID_Y; y++)
        {
            //se existirem aliens no limite direito
            if (spaceinvaders.grid[SPACEINVADERS_GRID_X-1][y] == 2)
            {
                //alteram o seu movimento, primeiro para baixo e depois para a esquerda
                spaceinvaders.aliens_move_right = false;
                spaceinvaders.aliens_move_left = true;
                spaceinvaders.aliens_move_down = true;
                break;
            }
        }
    }
}

//movimenta a nave consoante o input do jogador
void moveShip(bool left, bool right)
{
    if(right)//mover a nave para a direita
    {
        if(spaceinvaders.grid[SPACEINVADERS_GRID_X-1][SPACEINVADERS_GRID_Y-1] != 1) //se a
        nave não estiver o mais à direita possível
        {
            for (i = SPACEINVADERS_GRID_X - 1; i >= 0; i--) //encontrar todos os 1s que
            representam a posição do triângulo da nave
            {
                for(j = SPACEINVADERS_GRID_Y-1; j >= SPACEINVADERS_GRID_Y-3; j--)
                {
                    if(spaceinvaders.grid[i][j] == 1) // o 1 representa a posição do triângulo
                    da nave
                    {
                        if(spaceinvaders.grid[i+1][j] == 3)
                        {
                            spaceinvaders.over = true;
                        }
                        spaceinvaders.grid[i+1][j] = 1; // mudar a posição do 1 da nave para a
                        direita
                        spaceinvaders.grid[i][j] = 0;
                    }
                }
            }
        }
    }
    else if(left) // mover a nave para a esquerda
    {
        if(spaceinvaders.grid[0][SPACEINVADERS_GRID_Y-1] != 1) // se a nave não estiver o mais à
        esquerda possível
    }

```

```

{
  for (i = 0; i < SPACEINVADERS_GRID_X; i++) // encontrar todos os 1s que representam a
    posição do triângulo da nave
  {
    for(j = SPACEINVADERS_GRID_Y-1; j >= SPACEINVADERS_GRID_Y-3; j--)
    {
      if(spaceinvaders.grid[i][j]==1)
      {
        if(spaceinvaders.grid[i-1][j] == 3)
        {
          spaceinvaders.over = true;
        }
        spaceinvaders.grid[i-1][j] = 1; // mudar a posição do 1 da nave para a
        esquerda
        spaceinvaders.grid[i][j] = 0;
      }
    }
  }
}

//função usada para desenhar o jogo no lcd
void drawSpace()
{
  //tamanho dos blocos
  int block_size = WIDTH/SPACEINVADERS_GRID_X;

  display.clearDisplay();

  for (i = 0; i < SPACEINVADERS_GRID_X; i++)
  {
    //nave (um triângulo se block_size for impar ou dois triângulos adjacentes de block_size
    for par)
    if(i > 0 && i < SPACEINVADERS_GRID_X - 1 && spaceinvaders.grid[i-1][SPACEINVADERS_GRID_Y
    - 1] == 1 && spaceinvaders.grid[i+1][SPACEINVADERS_GRID_Y - 1] == 1)
    {
      if(block_size % 2 != 0)
        display.fillTriangle((i - 1)*block_size, (SPACEINVADERS_GRID_Y)*block_size-1,
        (i+2)*block_size-1, SPACEINVADERS_GRID_Y*block_size-1, (int)((i+0.5)*block_size),
        (SPACEINVADERS_GRID_Y-2)*block_size, BLACK);
      else
      {
        display.fillTriangle((i - 1)*block_size, (SPACEINVADERS_GRID_Y)*block_size-1,
        (int)((i+0.5)*block_size)-1, SPACEINVADERS_GRID_Y*block_size-1,
        (int)((i+0.5)*block_size)-1, (SPACEINVADERS_GRID_Y-2)*block_size, BLACK);
        display.fillTriangle((i+2)*block_size-1, SPACEINVADERS_GRID_Y*block_size-1,
        (int)((i+0.5)*block_size), SPACEINVADERS_GRID_Y*block_size-1,
        (int)((i+0.5)*block_size), (SPACEINVADERS_GRID_Y-2)*block_size, BLACK);
      }
    }

    for (j = 0; j < SPACEINVADERS_GRID_Y; j++)
    {
      //aliens (retangulos)
      if (spaceinvaders.grid[i][j] == 2)
      {
        display.drawRect(i*block_size,j*block_size ,block_size, block_size, BLACK);
      }
      //balas dos aliens (duas linhas verticais nos extremos do "bloco")
      if(spaceinvaders.grid[i][j] == 4)
      {
        display.writeFastVLine(i*block_size, j*block_size, block_size, BLACK);
        display.writeFastVLine((i+1)*block_size - 1,j*block_size ,block_size, BLACK);
      }
      //balas da nave (uma linha vertical no centro do "bloco" se block_size for impar ou
      duas se for par)
      if(spaceinvaders.grid[i][j] == 3)
      {
        if(block_size % 2 != 0)

```

```

        display.writeFastVLine((i+0.5)*block_size, j*block_size, block_size, BLACK);
    else
    {
        display.writeFastVLine((i+0.5)*block_size, j*block_size, block_size, BLACK);
        display.writeFastVLine((i+0.5)*block_size - 1, j*block_size, block_size, BLACK);
    }

}

}

//pontuação
display.setTextSize(1);
display.setTextColor(BLACK, WHITE);
display.setCursor(0, 0);
display.print(spaceinvaders.score);
}

//outras funções-----

//verifica se o botão esquerdo foi premido
bool checkLeftBtnPress()
{
    bool left_btn_state = digitalRead(LEFT_BTN_PIN);
    if(left_btn_state == HIGH && left_btn_state_prev == LOW)
    {
        left_btn_state_prev = left_btn_state;
        return true;
    }
    else
    {
        left_btn_state_prev = left_btn_state;
        return false;
    }
}

//verifica se o botão direito foi premido
bool checkRightBtnPress()
{
    bool right_btn_state = digitalRead(RIGHT_BTN_PIN);
    if(right_btn_state == HIGH && right_btn_state_prev == LOW)
    {
        right_btn_state_prev = right_btn_state;
        return true;
    }
    else
    {
        right_btn_state_prev = right_btn_state;
        return false;
    }
}

//verifica se o botão vermelho foi premido
bool checkRedBtnPress()
{
    bool red_btn_state = digitalRead(RED_BTN_PIN);
    if(red_btn_state == HIGH && red_btn_state_prev == LOW)
    {
        red_btn_state_prev = red_btn_state;
        return true;
    }
    else
    {
        red_btn_state_prev = red_btn_state;
        return false;
    }
}

//verifica se o botão verde foi premido

```

```
bool checkGreenBtnPress()
{
    bool green_btn_state = digitalRead(GREEN_BTN_PIN);
    if(green_btn_state == HIGH && green_btn_state_prev == LOW)
    {
        green_btn_state_prev = green_btn_state;
        return true;
    }
    else
    {
        green_btn_state_prev = green_btn_state;
        return false;
    }
}

//verifica se o botão verde foi solto
bool checkGreenBtnUnpress()
{
    bool green_btn_state = digitalRead(GREEN_BTN_PIN);
    if(green_btn_state == LOW && green_btn_state_prev == HIGH)
    {
        green_btn_state_prev = green_btn_state;
        return true;
    }
    else
    {
        green_btn_state_prev = green_btn_state;
        return false;
    }
}

//atualiza o delay dos jogos
void updateDelay()
{
    snake.game_delay = -113*difficulty + 600;
    breakout.game_delay = -33*difficulty + 200;
    tetris.game_delay = -113*difficulty + 800;
}

//mostra a mensagem de fim de jogo, pontuação e pontuação máxima
void gameOverText(int score, int max_score, bool won)
{
    display.clearDisplay();
    display.setTextSize(1);
    display.setTextColor(WHITE, BLACK);
    display.setCursor(10, 10);
    if(won)
        display.print("VITORIA!");
    else
        display.print("GAME OVER");
    display.setTextColor(BLACK, WHITE);
    display.setCursor(1, 25);
    display.print("Pontuacao: ");
    display.setCursor(60, 25);
    display.print(score);
    display.setCursor(1, 35);
    display.print("Pont.Max.: ");
    display.setCursor(62, 35);
    display.print(max_score);
}

//mensagem de jogo em pausa
void gamePausedText()
{
    display.setTextSize(1);
    display.setTextColor(WHITE, BLACK);
    display.setCursor(23, 20);
    display.print("PAUSED");
}
```

```
//função chamada quando a consola está em stand-by e a interrupção é acionada
void wakeUp()
{
    //a função não faz nada, serve apenas para sair do modo powerdown
}
```