ARDUBOY CLONE: Vídeo-game portátil

Equipe: Andrew C. Dellamea, Felipe A. Santana

Descrição do Projeto

O projeto foi desenvolvido a fim de expandir os conhecimentos na programação de Microcontroladores, desenvolvimento de jogos, e auxiliando a iniciativa open-source. O projeto envolveu a produção de um vídeo-game portátil, utilizando a biblioteca SSD1306, sendo programado dois jogos de autoria própria, sendo eles uma versão própria do jogo Pong e um jogo de batalha no espaço, inspirado em Star Wars, na qual foi nomeado como Space.

Lista de Materiais utilizados:

- 1 Arduino Uno
- 1 Cabo USB para Arduino
- 1 Display OLED 128x64px - 0.96" - 4 Pin - Branco
- 3 Botões táteis 6mm x 6mm
- 1 KC-1206 KC-1201 Passive Buzzer 12*9.5
- Cabos jumper
- 1 Protoboard de 400 furos

Tabela de orçamento do projeto

Componente	Preço (R$)
1 Arduino Uno + Cabo USB para Arduino	130,00
1 Display OLED 128x64px - 0.96" - 4 Pin - Branco	60,00
3 Botões táteis 6mm x 6mm	00,60
1 KC-1206 KC-1201 Passive Buzzer 12*9.5	03,00
Cabos jumper (Pack com 40)	09,00
Protoboard de 400 furos	14,00
Total	**216,60**

Aplicações do Projeto

Este projeto pode ser utilizado como sistema lúdico para ensino de microcontroladores, segundo SANTOS (2021) a gamificação como estratégia metodologia é capaz de facilitar o processo de entendimento e ensino de lógica para programação, tornando-a mais atrativa.

Objetivos

O presente projeto tem por finalidade, além de ampliar os conhecimentos sobre programação de microcontroladores, promover um exemplo de como seria possível desenvolver jogos na linguagem c para o microcontrolador ATMEL 328p, visto que poucos exemplos são encontrados pela web.

Objetivos Específicos

- Oferecer um código com implementação totalmente em c para controle do display SSD1306.
- Aplicar o que foi visto durante as aulas da componente curricular de Microcontroladores teórica e experimental
- Ampliar a capacidade de resolução de problemas

Circuito

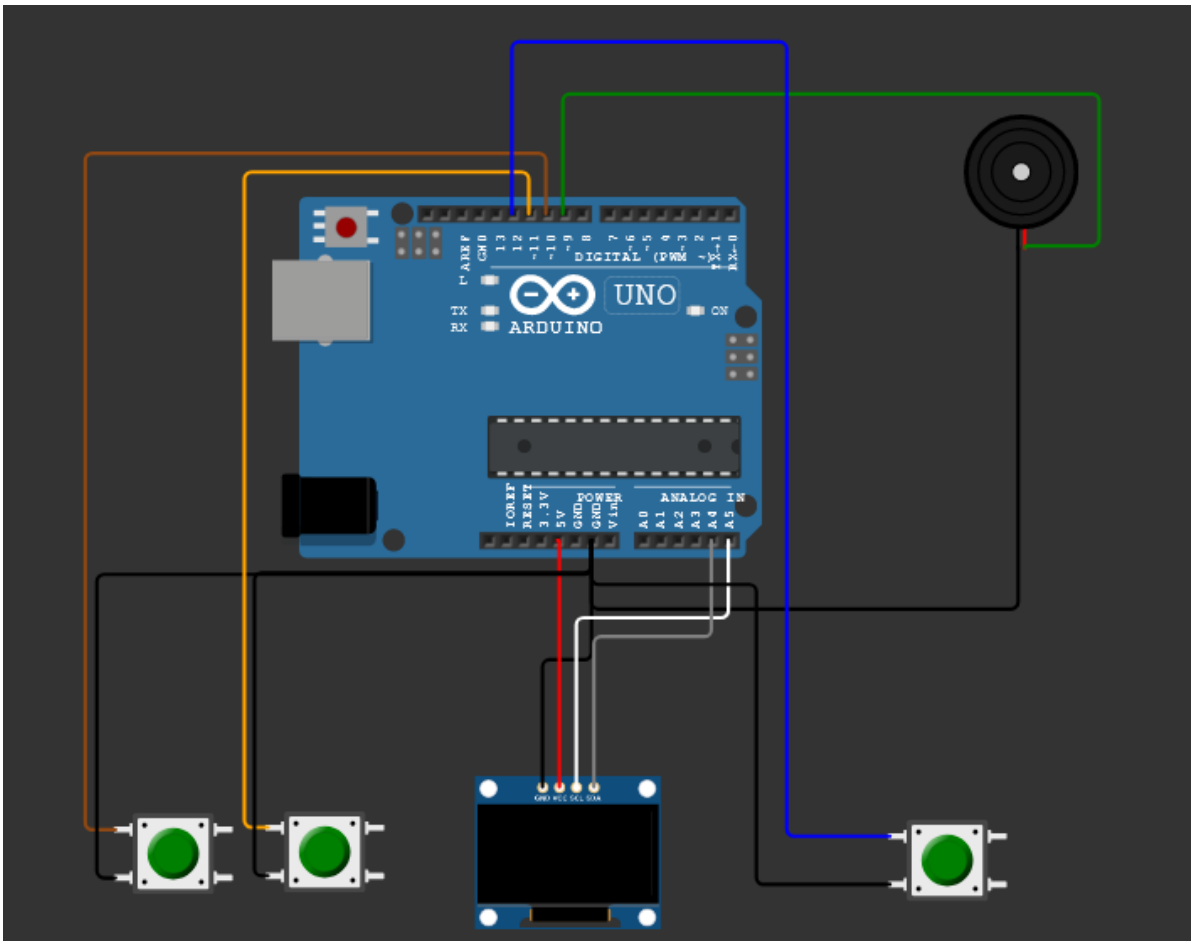

Figura 1: Esquemático do Projeto

Fotos

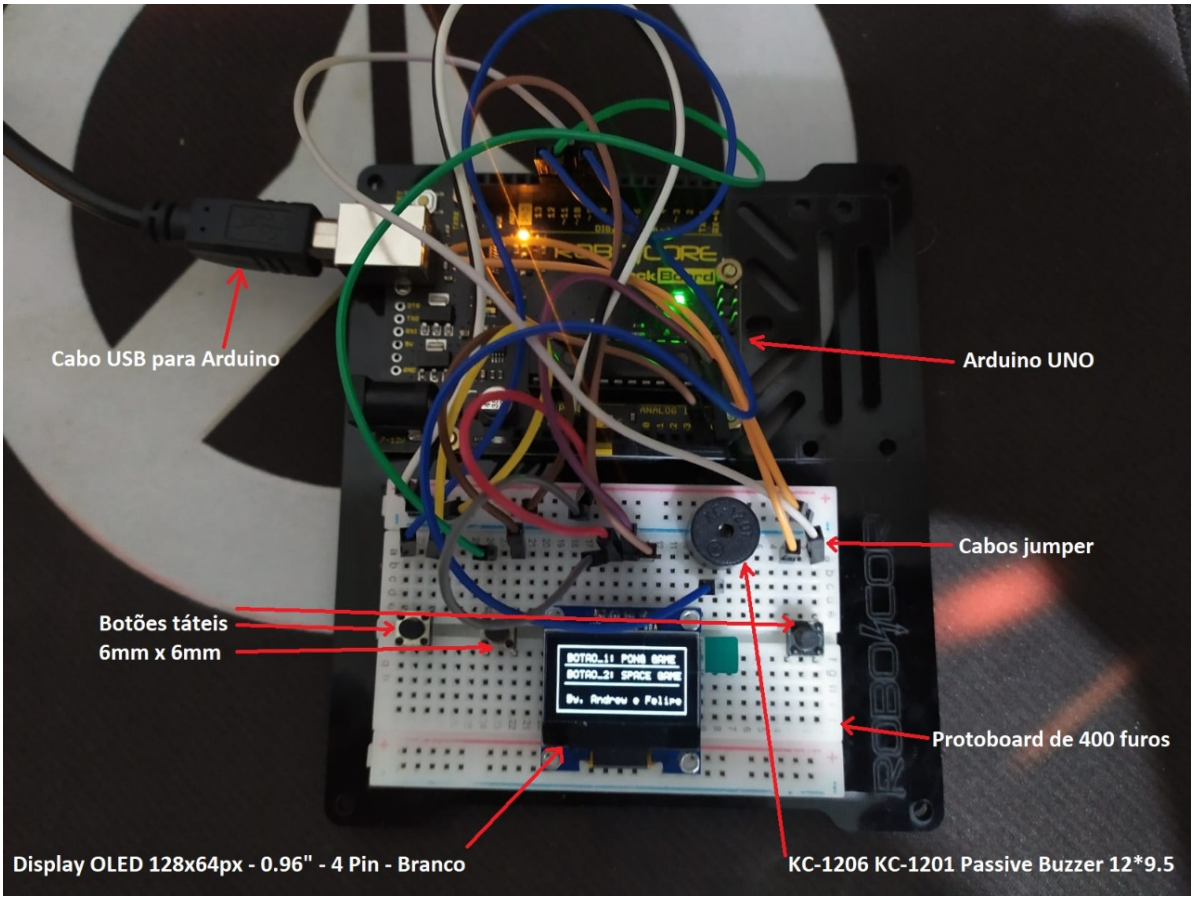

Figura 2: Projeto montado

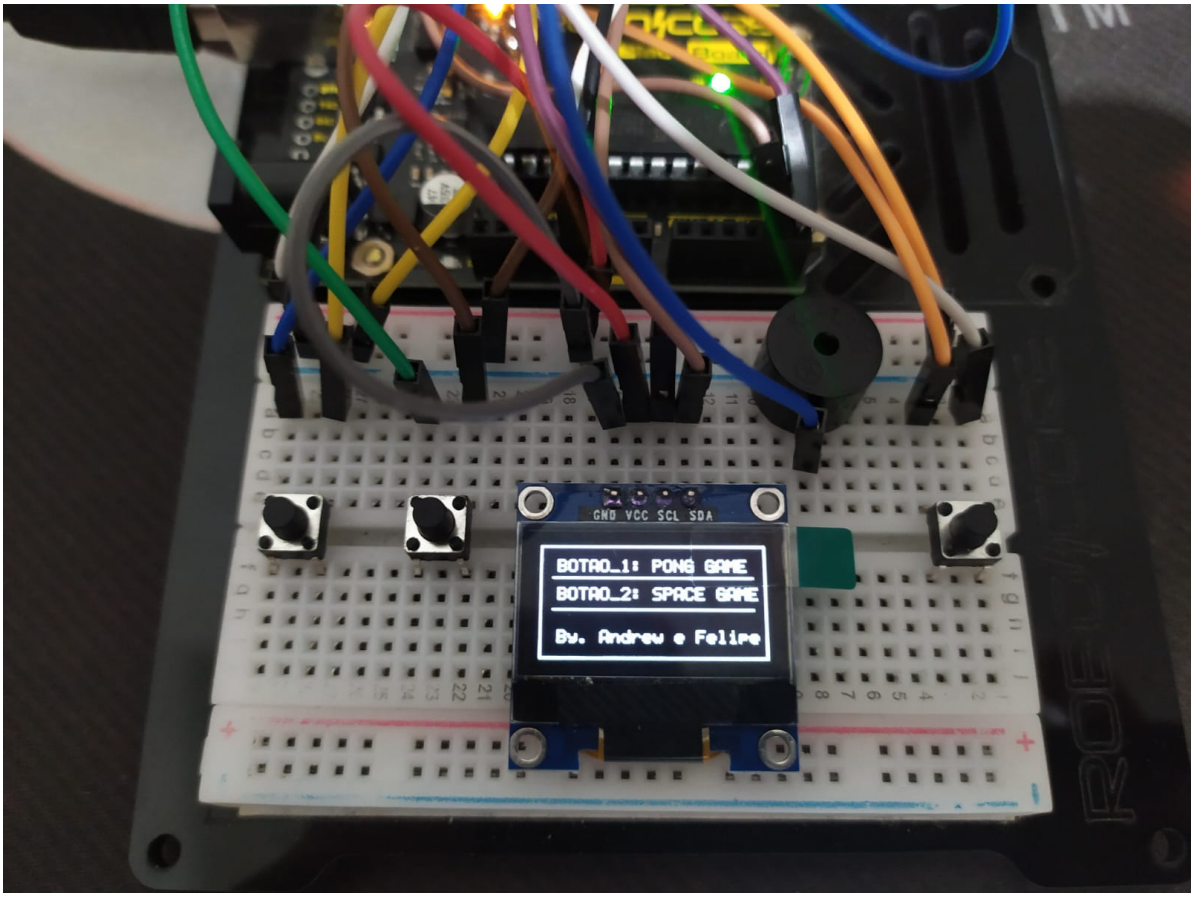

Figura 3: Menu Principal

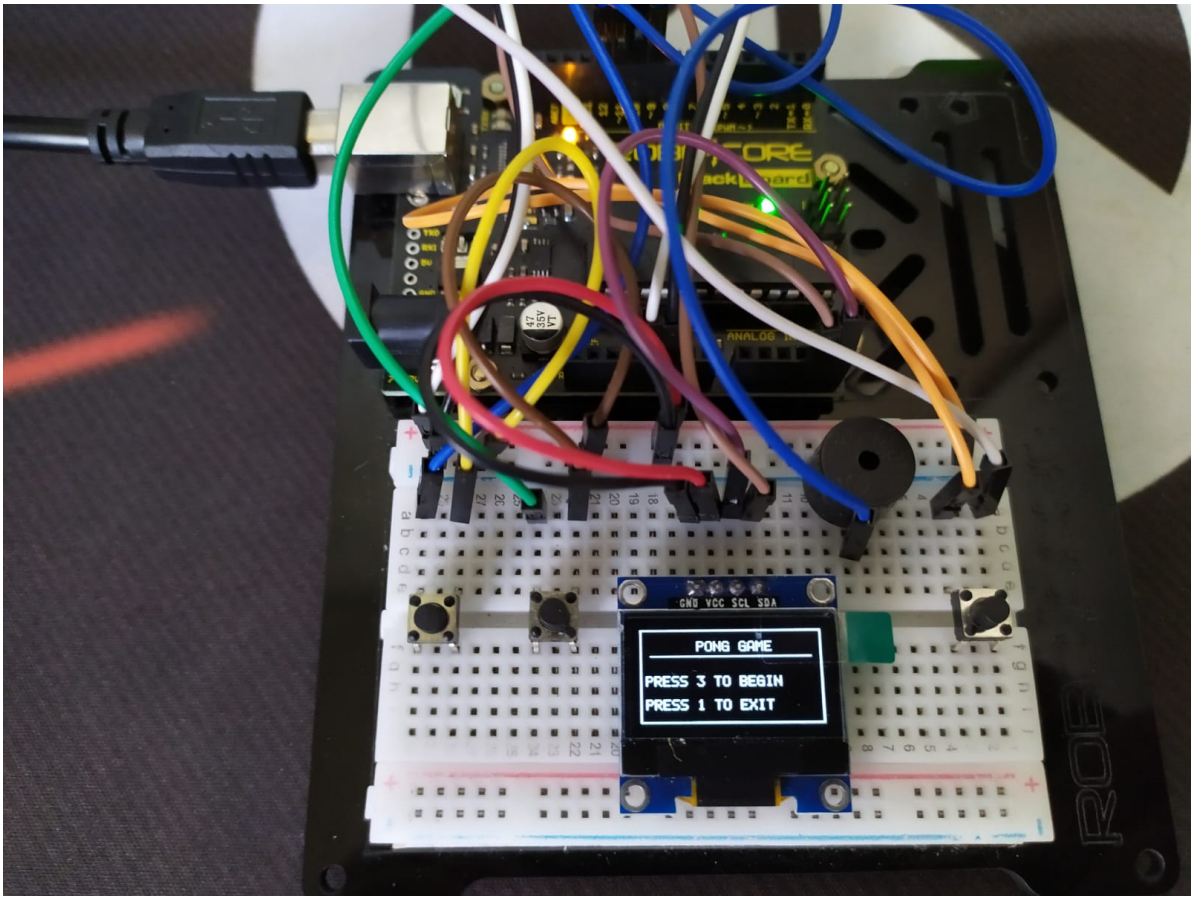

Figura 4: Menu Principal PONG

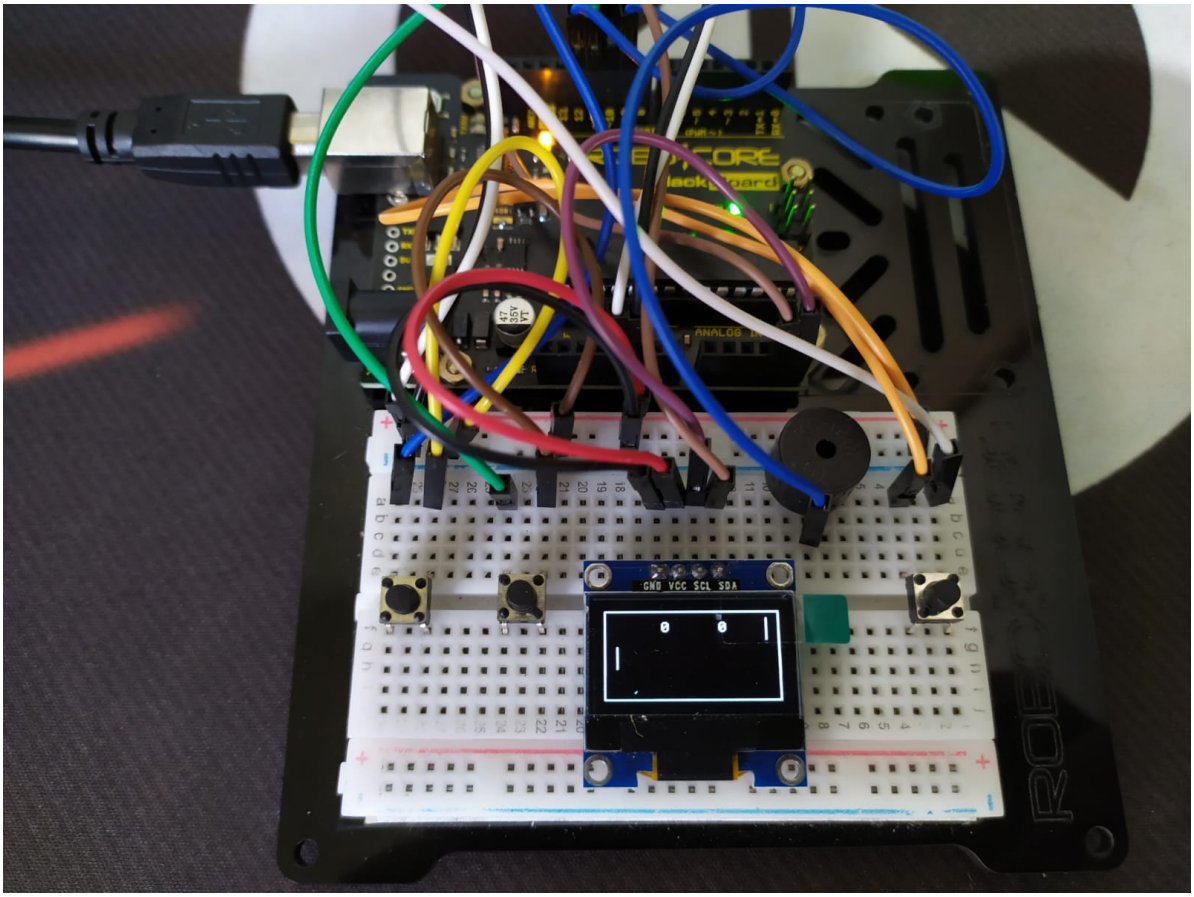

Figura 5: Jogo PONG

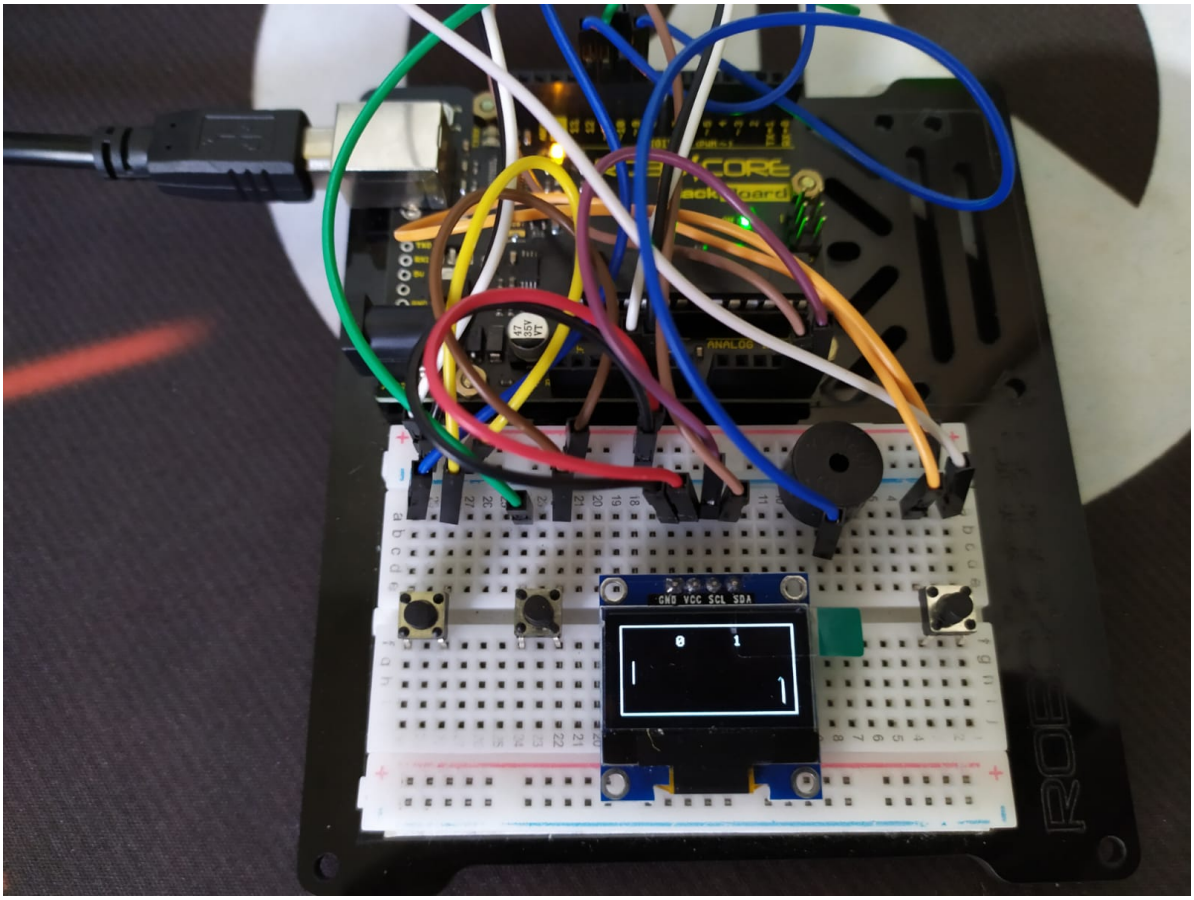

Figura 6: Atualização do Placar

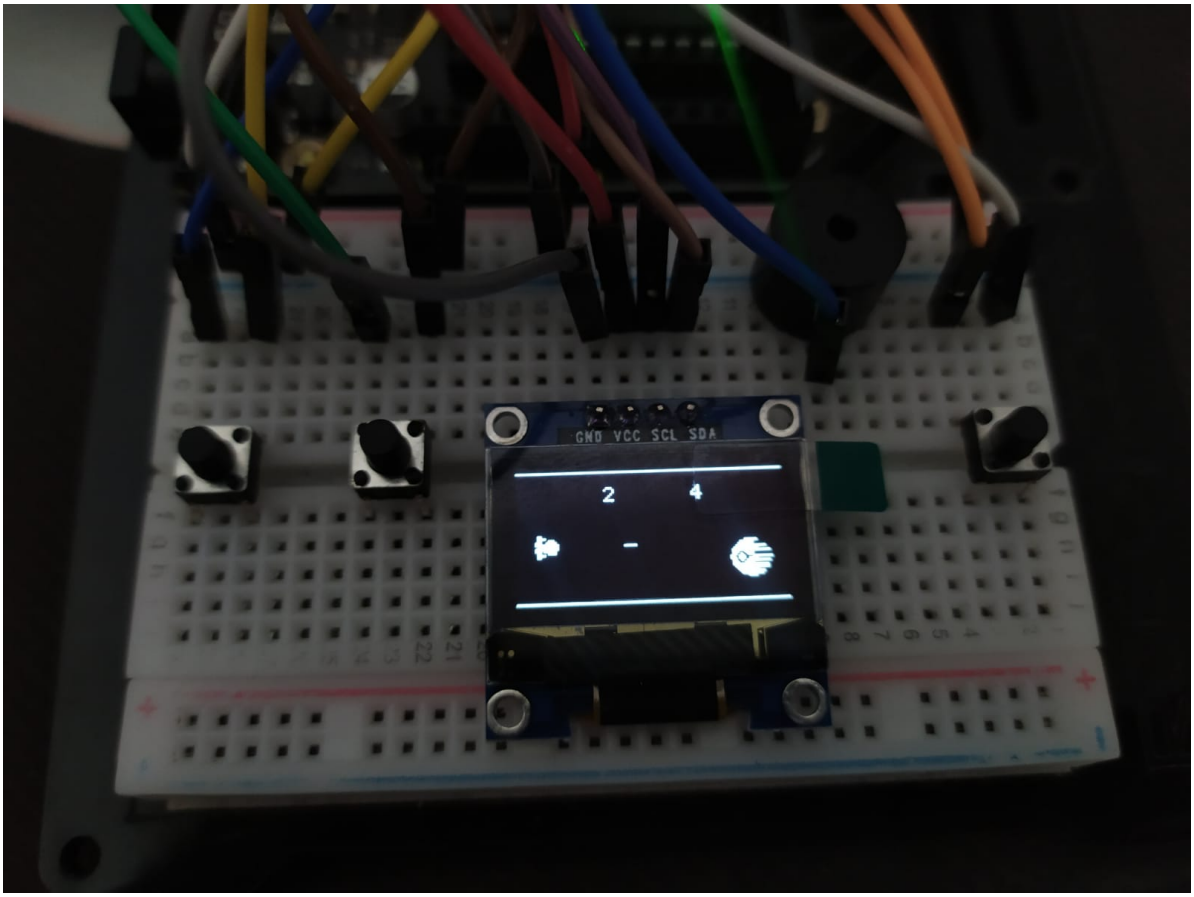

Figura 7: Jogo Space

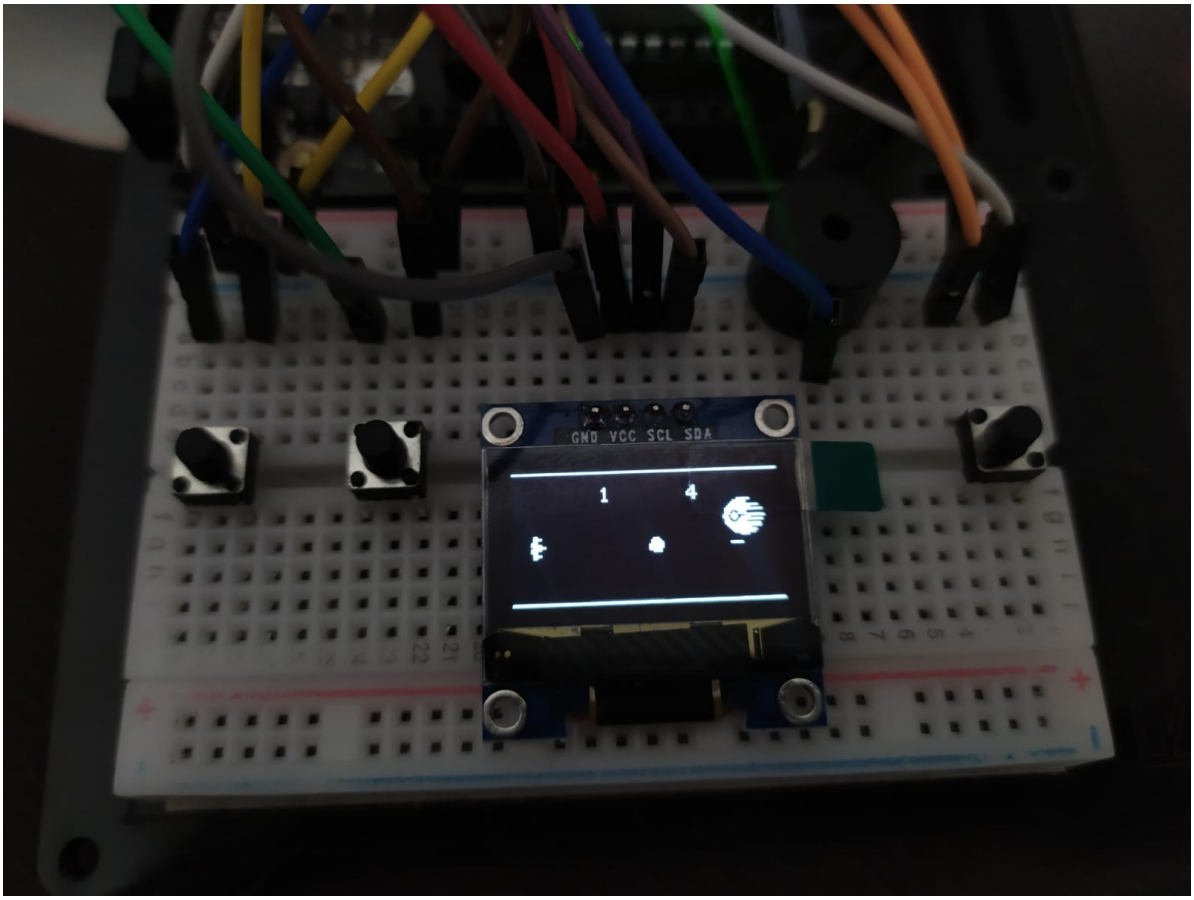

Figura 8: Atualização da vida

Explicação do Código

Para funcionamento do display SSD1306 128x64px foi utilizada uma biblioteca com implementações de fontes, atualização de pixels e sincronização de buffers. Esta biblioteca está disponível **aqui** (Matiasus/SSD1306 2021). Está seção é destinada a discutir as estratégias e funções que foram utilizadas para completar o projeto do Arduboy. Os códigos comentados estão presentes na integra no apêndice A.

Arquivo main.c

O programa main tem o objetivo de definir o endereço de inicialização do display e passa-lo para a função menu, visto que este endereço não pode ser declarado mais de uma vez.

```
#include <avr/io.h>
#include <avr/pgmspace.h>
#include <util/delay.h>
#include <menu.h>
#include <ssd1306.h>
//Definicoes de macros
#define set_bit(Y,bit_x) (Y|=(1<<bit_x)) //ativa bit
#define clr_bit(Y,bit_x) (Y&=~(1<<bit_x)) //limpa bit
uint8_t addr = SSD1306_ADDRESS; //endereço da tela para recebimento de comandos
int main() {
    while(1) {
        menu(addr);
        _delay_ms(1000);
    }
}
```

Arquivo menu.c

Para definição do menu, foi criada uma função chamada `draw_Screen()` para desenhar o menu. Nela é passado o endereço da tela como argumento a fim de utilizar as funções da biblioteca SSD1306 para desenhar linhas e escrever strings para definir uma interface simples para o usuário, descrevendo o que cada botão irá ativar.

```
void draw_Screen(uint8_t addr)
{
    SSD1306_DrawLine(0, MAX_X, 0, 0);
    SSD1306_DrawLine(0, MAX_X, 63, 63);
    SSD1306_DrawLine(0, 0, 0, MAX_Y);
    SSD1306_DrawLine(127, 127, 0, MAX_Y);
    SSD1306_SetPosition(10, 1);
    SSD1306_DrawString("BOTAO_1: PONG GAME");
    SSD1306_DrawLine(7, 120, 18, 18);
    SSD1306_SetPosition(10, 3);
    SSD1306_DrawString("BOTAO_2: SPACE GAME");
    SSD1306_DrawLine(7, 120, 36, 36);
    SSD1306_SetPosition(10, 6);
    SSD1306_DrawString("By. Andrew e Felipe");
    SSD1306_UpdateScreen(addr);
}
```

Outra função criada é a própria `menu()`, que chama a `draw_Screen` e define o jogo que será acionado em cada botão.

pong.c

O primeiro jogo a ser implementado foi o famoso pong, nele o jogador assume o papel de um tenista de mesa e precisa impedir que seu oponente faça pontos antes de si mesmo. Algumas definições foram adicionadas e são explicadas no código.

Antes de iniciar o jogo, algumas definições foram feitas, como os pinos utilizados para os botões de controle, sendo eles o **PB1** para produção de sons, o **PB2** como botão seta para cima, **PB3** como botão seta para baixo e o **PB4** como botão de ação, que neste jogo é utilizado para iniciar o mesmo. Algumas definições quando o funcionamento do buzzer utilizado também são encontrados neste bloco. Ao tentar utilizar o tipo booleano `bool`, o compilador demonstrou um erro, descrevendo que o mesmo não existia, portanto, foram definidas os valores de `true = 1`, `false = 0`, `nada = -1` em uma tentativa de contornar o problema. Nesta mesma seção, foi definido o tamanho da plataforma como sendo 15 pixels na constante `tam_plat`.

```
#define BOTAO_UP PB4
#define BOTAO_DOWN PB3
#define SELECT PB2
// define configuracoes som
#define som PB1                // pino OC1A para saida do sinal
#define d_inic 4               // valor inicial de duracao da nota musical
#define o_inic 5               // valor inicial da oitava
#define b 192                 /*o nr de batidas indica a velocidade da musica (alterar
para mudar a velocidade), maior = mais rapido*/
#define t_min (7500 / b) * 10 // tempo minimo para formar o tempo base das notas
musicais (1/32)
// define configuracoes jogo
#define true 1
#define false 0
#define nada -1
```

```
#define tam_plat 15
```

Para controlar a lógica do jogo, foram definidas variáveis que armazenam as posições x e y inicial do jogador 1 (`player1_x` e `player1_y`), do jogador 2 (`player2_x` e `player2_y`) e da bola (`ball_x` e `ball_y`). Para armazenar o placar do jogo, foram declaradas as variáveis `score_pl_1` e `score_pl_2` para os pontos do jogador um e 2 respectivamente. `invert_ball_y` e `invert_ball_y` são utilizados para definir se os valores x e y da bola devem ser incrementados (caso `true`) ou decrementados (caso `false`). `start_game` foi utilizado, para definir se o jogo seria iniciado, ou voltaria para o menu principal. A `dificuldade` define a posição em x que a I.A. do jogador numero dois irá ativar. `musica` armazena o som tocado quando a bola colide com os jogadores. `ball_direction_preset` controla a pseudo-aleatoriedade da bola, com algumas direções predefinidas.

```
// variaveis para o pong
int8_t player1_x = 9;
int8_t player1_y = 26;
int8_t player2_x = 118;
int8_t player2_y = 26;
int8_t ball_x = 62;
int8_t ball_y = 30;
int8_t score_pl_1 = 0;
int8_t score_pl_2 = 0;
int8_t invert_ball_y = true;
int8_t invert_ball_x = true;
int8_t start_game = false;
int8_t ball_direction_preset = 1;
uint8_t dificuldade = 90;
// musica
const char musica[] PROGMEM = {"8f,8f"};
```

As funções de desenho e controle da lógica do jogo são apresentadas abaixo:

```
// functions for pong
void draw_Box(void);  //desenha bordas
void draw_player_1(void); //desenha jogador 1
void draw_player_2(void); //desenha jogador 2 e controla a I.A.
void drawBall(void);  //atualiza a posicao e desenha a bola
void update_ball_x(void); //atualiza a posicao da bola baseado no invert_ball_x
void update_ball_y(void); //atualiza a posicao da bola baseado no invert_ball_y
void invert_ball_direction(void); // determina se deve inverter as variaveis
Invert_ball
void draw_match_numbers(void); // desenha na tela o placar
void game_over_screen(uint8_t); //desenha a tela de fim de jogo e redefine as
variaveis do jogo
void init_screen(uint8_t); // apresenta tela de inicio do jogo
void test_colision(uint8_t, uint8_t, int8_t); //testa se a bola colidiu com alguma
jogador
void toca_musica(const char *); // toca o som da colisao
void pong_init(uint8_t); // funcao principal
```

Para limitar a quantidade de código apresentado, serão explicados as principais funções que fazem o jogo funcionar sendo elas: `invert_ball_direction()` , `test_colision()` , `draw_player_2()` e `pong_init()` . O funcionamento está descrito nos códigos.

`invert_ball_direction()` : verifica se a bola colidiu com uma parede horizontal, e chama a função `test_colision()` para verificar se houve colisão com os jogadores.

```
// inverte direcao da bola
```

```
void invert_ball_direction()
{
  // verifica se a bola bateu em alguma das paredes horizontais
  if ((ball_y == 1) || (ball_y == 62))
  {
    // se sim, inverte sua direcao em y
    invert_ball_y = !invert_ball_y;
  }
  // verifica se a bola chegou na coluna dos players
  test_colision(player1_x, player1_y,1);
  test_colision(player2_x, player2_y,-1);
  // testa se a bola ultrapassou os players, ou seja, nao houve colisao
  if ((ball_x == 1) || (ball_x == 126))
  {
    if (ball_x == 1) // se ultrapassar o jogador 1
    {
      score_pl_2++; //adiciona ponto para o jogador 2
    }
    else if (ball_x == 126) //se ultrapassar o jogador 2
    {
      score_pl_1++;  //adiciona ponto ao jogador 1
    }
    // ao fazer gol, reseta a posicao da bola para o centro da tela
    ball_x = MAX_X / 2;
    ball_y = MAX_Y / 2;

 switch (ball_direction_preset) // controla a pseudo - aleatoriedade da bola
    {
    case 1: //nao inverte a direcao da bola
      invert_ball_x = invert_ball_x;
      invert_ball_y = invert_ball_y;
      break;
    case 2: //inverte apenas a direcao de x
      invert_ball_x = !invert_ball_x;
      invert_ball_y = invert_ball_y;
      break;
    case 3: // inverte a direcao de y apenas
      invert_ball_x = invert_ball_x;
      invert_ball_y = !invert_ball_y;
      break;
    case 4: // inverte totalente a direcao
      invert_ball_x = !invert_ball_x;
      invert_ball_y = !invert_ball_y;
      break;
    }
    ball_direction_preset++; // muda o preset da bola
  }
  update_ball_x(); //atualiza se deve incrementar ou decrementar, baseado nas mudanças
feitas nos testes de colisao.
  update_ball_y();
}
```

`test_colision()` : recebe a posição x e y do player que deseja-se verificar, e o valor de ajuste para animação da colisão. O jogador numero 1 recebe valor +1 para que o teste ocorra um pixel a sua frente. Da mesma forma, o jogador numero 2 recebe este ajuste, porem com valor oposto, para que a colisão ocorra a cima da barra, ou invés de dentro dela.

```
void test_colision(uint8_t player_x, uint8_t player_y, int8_t ajuste)
{
  int player_y_end = (tam_plat + player_y); // calcula o valor de y final para o
jogador
  if (player_x + ajuste == ball_x) // define o pixel o qual o teste começa
  {
    uint8_t fragment_of_paddle = tam_plat / 3; // calcula o tamanho das seções do
jogador, para que a physica da bola seja diferente entre a parte superior, central e
inferior
    uint8_t lim_part_1 = player_y_end - (fragment_of_paddle * 2);//tamanho de 2/3 da
barra
    uint8_t lim_part_2 = player_y_end - (fragment_of_paddle); //tamanho de 1/3 da
barra
    if ((ball_y >= player_y) && (ball_y <= lim_part_1)) // testa para ver se a bola
colidiu na parte superior do jogador
    {
        //se sim, inverte x e diminui y, a bola sobe na diagonal superior
      invert_ball_x = !invert_ball_x;
      invert_ball_y = false;
      toca_musica(musica); //reproduz som de colisao
    }
    else if ((ball_y >= lim_part_1) && (ball_y <= lim_part_2)) //testa se a bola
colidiu na parte central do jogador
    {//se sim, inverte x e anula y, a bola deve seguir reta neste caso
      invert_ball_x = !invert_ball_x;
      invert_ball_y = nada;
      toca_musica(musica);
    }
    else if ((ball_y >= lim_part_2) && (ball_y <= player_y_end))// testa se a bola
colidiu na parte inferior do jogador
    { //se sim, inverte x e aumenta y, a bola sobe na diagonal inferior
      invert_ball_x = !invert_ball_x;
      invert_ball_y = true;
      toca_musica(musica);
    }

  } /*
    if (ball_y >= player_y && ball_y <= player_y_end)
    {
      invert_ball_x = !invert_ball_x;
       toca_musica(musica);
    }*/
}
```

`draw_player_2()` : Antes de desenhar o jogador 2, atualiza sua posição em relação a bola, para criar uma inteligência artificial e proporcionar algum desafio ao usuário.

```
// desenha a barra do jogador numero 2
void draw_player_2()
{
  if (ball_x > dificuldade) // controla I.A. do jogador numero 2,
  {                          //começa a se mover
                                //quando a bola_x chega em 74.

    if ((ball_y < player2_y) && (player2_y > 1))  //se a posiçao y da bola for
    {                                             //inferior ao y do jogador 2,
                                                  //eleva a barra
      player2_y--;
```

```
    }
    if ((ball_y > player2_y) && ((player2_y + tam_plat) < 62)) //se a posicao y
    {                                                          //da bola for superior
       player2_y++;                                          //a do jogador 2, rebaixa
                                                           //a barra.
    }
  }

  SSD1306_DrawLine(player2_x, player2_x, (player2_y), (player2_y + tam_plat));
}
```

`pong_init()` : Controla o fluxo do jogo, e é a função chamada no **menu.c**.

```
void pong_init(uint8_t address)
{

  int addr = address; // armazena o endereco de comandos da tela
  DDRB = 0b00000000;  // portas 2,1 e 0 como entrada;
  PORTB = 0b11111111; // habilita pullup para entradas
  // inicia pong
  SSD1306_ClearScreen();
  // INICIA SETUP DO JOGO
  while (1)
  {
    init_screen(addr);// desenha a tela de inicio do jogo

    if (!tst_bit(PINB, BOTAO_UP)) // volta para o menu principal
    {
      break;
    }
    if (!tst_bit(PINB, SELECT)) // inicia o jogo
    {
      start_game = true;
    }

    if (start_game) // o jogo
    {
      while (1)
      {
        SSD1306_ClearScreen(); //limpa a tela
        drawBox(); //desenha as bordas
        draw_match_numbers(); //desenha o placar
        draw_player_1(); //desenha o jogador 1
        draw_player_2(); //desenha o jogador 2
        drawBall(); // desenha a bola

        if (!tst_bit(PINB, BOTAO_DOWN) && ((player1_y + tam_plat) < 62))
        {//testa para ver se o botao 2 mudou de estado
          player1_y++; //se sim incrementa o valor de y do jogador 1
        }
        if (!tst_bit(PINB, BOTAO_UP) && (player1_y > 1))
        {//testa para ver se o botão 1 mudou de estado
          player1_y--;//se sim decrementa o valor do y do jogador 1
        }
        if ((score_pl_1 == 3) || (score_pl_2 == 3))
        {//se o placar de qualquer um dos jogadores chegar a 3, inicia o processo de
fim de jogo
          start_game = 0; //anula o estado do jogo
```

```
          break; //sai do loop
        }
        SSD1306_UpdateScreen(addr);//atualiza a tela com os objetos desenhados
      }
      if (start_game == 0)//testa se deve ativar a tela de fim de jogo
      {
        game_over_screen(addr);//caso o jogador 1 tenha ganho, apresenta na tela, se
nao, apresenta que foi o jogador numero 2.
        break;// finaliza o jogo e volta ao menu principal de jogos
      }
    }
  }
}
```

Para que a função `draw_match_numbers()` funcionasse, foram necessárias algumas modificações em **ssd1306.c**. Foram modificadas algumas linhas na função `SSD1306_DrawChar()` para que fosse possível desenhar um char em uma posição especifica. Para isso, a posição x,y e o valor do que deve ser apresentado são passados como argumentos a função criada `SSD1306_DrawCharAt()`.

```
uint8_t SSD1306_DrawCharAt(uint8_t x, uint8_t y, int a)
{
  char num = 'a'; //declara uma variavel char
  uint8_t i = 0;
  uint8_t page = 0;
  // uint8_t pixel = 0;

  // if out of range
  if ((x > MAX_X) && (y > MAX_Y)) // verifica se o pixel nao esta fora dos limites
  {
    // out of range
    return SSD1306_ERROR;
  }
  // find page (y / 8)
  page = y >> 3;
  // which pixel (y % 8)
  // pixel = 1 << (y - (page << 3));
  // update counter
  _counter = x + (page << 7);
  // save pixel
  //define qual char deve ser apresentado
  if (a == 0)
  {
    num = '0';
  }
  else if (a == 1)
  {
    num = '1';
  }
  else if (a == 2)
  {
    num = '2';
  }
  else if (a == 3)
  {
    num = '3';
  }
  else if (a == 4)
```

```
  {
    num = '4';
  }
  else if (a == 5)
  {
    num = '5';
  }

  while (i < CHARS_COLS_LENGTH)
  {
    cacheMemLcd[_counter++] = pgm_read_byte(&FONTS[num - 32][i++]);
  }
  // success
  return SSD1306_SUCCESS;
}
```

O segundo jogo a ser implementado foi o nomeado space, inspirado na franquia Star Wars, nele o jogador assume o papel de controlar a nave e precisa derrotar seu oponente, a "Death Star", antes que ela derrote sua nave. Foram feitas as mesmas definições utilizadas no jogo pong, apenas adicionando os tamanhos do tiro da nave 1, como sendo 5 pixels na constante `TAM_TIRO` e o tiro da nave 2, como sendo 7 pixels na constante `TAM_TIRO2`.

```
#define TAM_TIRO 5   //Tamanho do tiro da nave 1
#define TAM_TIRO2 7  //Tamanho do tiro da nave 2
```

Para controlar a lógica do jogo, foram definidas variáveis que armazenam as posições x e y inicial da nave 1 (`nave_x` e `nave_y`), nave 2 (`nave_x2` e `nave_y2`), tiro da nave 1 (`tiro_x` e `tiro_y`) e tiro da nave 2 (`tiro2_x` e `tiro2_y`), além do tamanho das duas naves (`tam_nave` e `tam_nave2`). Para armazenar o placar do jogo, foram declaradas as variáveis `score_nave1` e `score_nave2` para os pontos de vida do jogador 1 e 2 respectivamente. `tiro_control` e `tiro2_control` são utilizados para definir se os jogadores podem atirar (caso `true`) ou não (caso `false`). O `nave2_control` foi utilizado para definir o estado da nave 2, para que a mesma possa se movimentar para cima e para baixo. `start_game2` foi utilizado, para definir se o jogo seria iniciado, ou voltaria para o menu principal. `musica2` armazena o som tocado quando as naves são atingidas por um tiro.

```
// variaveis para o space
int start_game2 = false;

int nave_x = 9;
int nave_y = 26;
int tam_nave = 11;

int nave_x2 = 95;
int nave_y2 = 16;
int tam_nave2 = 32;

int tiro_y = 0;
int tiro_x = 0;

int tiro2_y = 0;
int tiro2_x = 0;

uint8_t tiro_control = false;
uint8_t tiro2_control = false;
```

```
uint8_t nave2_control = false;

int score_nave1 = 3;
int score_nave2 = 5;

const char musica2[] PROGMEM = {"8f,8f"}; //musica
```

Foram utilizados as variáveis `st_bitmap_player1`, `st_bitmap_player2` e `st_bitmap_tiro_player2` para definir os bitmaps, ou seja os pixels que devem ser ligados, para ser feito os desenhos da nave 1, 2 e o tiro da nave 2.

```
// Bitmaps
const uint8_t st_bitmap_player1[] =
    {
        0x06, 0x1e, 0x06, 0x07, 0x1f, 0x7e, 0x1f, 0x07, 0x06, 0x1e, 0x06};

const uint8_t st_bitmap_player2[] =
    {
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x80, 0x80,
        0x80, 0xc0, 0xc0, 0x80, 0x80, 0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xe0, 0xf8, 0xfc, 0xfe,
0xff, 0xff, 0xe3,
        0xc9, 0x1c, 0x9d, 0xc9, 0xe3, 0xff, 0xff, 0xfe, 0xfc, 0xf8, 0xc0, 0x00, 0x00,
0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x07, 0x01, 0x01, 0x7f,
0x07, 0xff, 0x01,
        0x01, 0xfe, 0x07, 0xff, 0x01, 0xff, 0xff, 0x07, 0x3f, 0x1f, 0x03, 0x00, 0x00,
0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x01, 0x00,
        0x00, 0x03, 0x00, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00};

const uint8_t st_bitmap_tiro_player2[] =
    {
        /* 00111000 */ 0x038,
        /* 01111100 */ 0x07c,
        /* 11111100 */ 0x0fc,
        /* 11111110 */ 0x0fe,
        /* 11111110 */ 0x0fe,
        /* 01111110 */ 0x07e,
        /* 01111000 */ 0x078};
```

As funções de desenho e controle da lógica do jogo são apresentadas abaixo:

```
// functions for space
void drawBox2(void); //desenha bordas
void draw_nave(int); //desenha nave 1
void draw_nave2(int); //desenha nave 2 e controla a I.A
void init_screen2(uint8_t); //apresenta a tela inicial do jogo
void space_init(uint8_t); //função principal
void draw_tiro(uint8_t, uint8_t); //desenha o tiro da nave 1 e atualiza a posicao e
condição para atirar
void draw_tiro2(uint8_t, uint8_t); //desenha o tiro da nave 2 e atualiza a posicao e
condição para atirar
void test_colision_nave1(uint8_t, uint8_t, uint8_t); //testa se o tiro da nave 2
colidiu com a nave 1
void test_colision_nave2(uint8_t, uint8_t, uint8_t); //testa se o tiro da nave 1
colidiu com a nave 2
void toca_musica2(const char *); //toca o som da colisão
void draw_match_numbers2(void); //desenha na tela a vida das naves
void game_over_screen2(uint8_t); //desenha a tela de fim de jogo e redefine as
variaveis do jogo
```

Para limitar a quantidade de código apresentado, serão explicados as principais funções que fazem o jogo funcionar sendo elas: `draw_nave2()`, `draw_tiro()`, `test_colision_nave1()` e `space_init()`. O funcionamento está descrito nos códigos.

`draw_nave2()`: Antes de desenhar a nave 2, atualiza sua posição, para a mesma se movimentar para cima e para baixo.

```
// desenha a nave 2
void draw_nave2(int addr)
{

    if (nave_y2 + 5 > 0 && nave2_control == false) // Se a posição y da nave for
superior  0, eleva a nave
    {                                           //É adicionado +5 para a nave parar
5 pixels antes da barra superior
        nave_y2--;                              // nave2_control para controle de
prioridade da nave seguir pra uma direção
    }                                           // false: para nave subir e true:
para nave descer.
    if (nave_y2 + 5 == 1)
    {
        nave2_control = true; // a nave chegou ao limite superior, então alterou o
controle para poder descer
    }
    if ((nave_y2 + tam_nave2 - 3) < 63 && nave2_control == true) // Se a posição y da
nave for inferior a 63, rebaixa a nave
    {                                                           //É adicionado -3
para a nave parar 3 pixels antes da barra inferior
        nave_y2++;                                              //+ o tam_nave2 para
o desenho não ultrapassar o limite
    }                                                           // nave2_control para
controle de prioridade da nave seguir pra uma direção
    if ((nave_y2 + tam_nave2 - 3) == 62)
    {
        nave2_control = false; // a nave chegou ao limite inferior, então alterou o
controle para poder subir
    }
```

```
    drawBitMap(nave_x2, nave_y2, st_bitmap_player2, tam_nave2, 4); // chama o metódo
para desenhar o bitmap (classe: SSD1306)
}
```

`draw_tiro()` : Antes de desenhar o tiro, atualiza sua posição, para o mesmo ser desenhado a partir da ponta da nave.

```
// desenha o tiro 1
void draw_tiro(uint8_t addr, uint8_t controle)
{
    //controle é iniciado como false
    if (!controle)
    {
        tiro_y = (nave_y + (tam_nave / 2)); //calcula para que o tiro seja disparado
do meio da nave
        tiro_x = (nave_x + 8); //calcula para que o tiro seja disparado da frente da
nave
    }
    //Quando o botão é precinado o controle é alterado para true
    if (controle)
    {
        if (tiro_x + TAM_TIRO == 110) //Verifica se o tiro chegou no limite da tela
        {
            tiro_control = false;     //Caso verdadeiro, apaga o desenho.
            _delay_ms(250);
        }
        SSD1306_DrawLine(tiro_x, (tiro_x + TAM_TIRO), tiro_y, tiro_y); // chama o
metódo para desenhar a linha do tiro (classe: SSD1306)
        tiro_x++; //incrementa em x para movimentar o desenho
    }
}
```

No `draw_tiro2` é feito o mesmo, sendo o desenho de acordo com o da nave 2, adicionando uma função para que a nave 2 dispare no centro da nave 1.

```
// desenha o tiro 2
void draw_tiro2(uint8_t addr, uint8_t controle2)
{
    // controle é iniciado como false
    if (!controle2)
    {
        tiro2_y = (nave_y2 + (tam_nave2 / 2)); // calcula para que o tiro seja
disparado do meio da nave
        tiro2_x = (nave_x2 - 2); // calcula para que o tiro seja disparado da frente
da nave
    }
    // Nave 2 atira no centro da nave 1
    if ((nave_y - 14) == nave_y2)
    {
        tiro2_control = true; //controle é alterado para true para o mesmo disparar
    }
    //controle é alterado para true quando a nave 1 estiver no centro da nave 2
    if (controle2)
    {
        if (tiro2_x == 7) // Verifica se o tiro chegou no limite da tela
        {
```

```
            tiro2_control = false; // Caso verdadeiro, apaga o desenho.
        }

        drawBitMap(tiro2_x, tiro2_y, st_bitmap_tiro_player2, TAM_TIRO2, 1); // chama o
metódo para desenhar o bitmap (classe: SSD1306)
        tiro2_x--;  // decrementa em x para movimentar o desenho da direita para
esquerda
    }
}
```

`test_colision_nave1()`: recebe a posição x e y da nave 1, e o valor de ajuste para animação da colisão. A nave 1 recebe valor +1 em x para que o teste ocorra um pixel a sua frente. Da mesma forma, a nave 2 recebe este ajuste no valor de +2 em x, por conta do seu bitmap do desenho, para que a colisão ocorra em cima do desenho ao invés de dentro ou antes do mesmo.

```
// testa se o tiro da nave 2 colidiu com a nave 1
void test_colision_nave1(uint8_t player_x, uint8_t player_y, uint8_t ajuste)
{
    int player_y_end = (tam_nave + player_y); // calcula o valor de y final para a
nave
    if (player_x + ajuste == tiro2_x)         // define o pixel o qual o teste começa
    {
        if (tiro2_y >= player_y && tiro2_y <= player_y_end) // testa para ver se o
tiro colidiu na nave 1
        {
            toca_musica2(musica2); // reproduz som de colisao
            score_nave1--;         // se, sim decrementa a vida da nave 1
        }
    }
}
```

O `test_colision_nave2()` possui mais um ajuste, diminuindo o valor de -2 em y, para corrigir a área da nave 2, por conta do seu bitmap.

`space_init()`: Controla o fluxo do jogo, e é a função chamada no **menu.c**.

```
void space_init(uint8_t address)
{

    int addr = address; // armazena o endereco de comandos da tela
    DDRB = 0b00000000;  // portas 2,1 e 0 como entrada;
    PORTB = 0b11111111; // habilita pullup para entradas
    // inicia space
    SSD1306_ClearScreen();
    // INICIA SETUP DO JOGO
    while (1)
    {
        init_screen2(addr);

        if (!tst_bit(PINB, BOTAO_UP)) // volta para menu principal
        {
            break;
        }
        if (!tst_bit(PINB, SELECT)) // inicia o jogo
        {
            start_game2 = true;
        }
```

```
        if (start_game2) // o jogo
        {
            while (1)
            {
                SSD1306_ClearScreen(); //limpa a tela
                drawBox2(); //desenha as bordas
                draw_nave(addr); //desenha a nave 1
                draw_nave2(addr); //desenha a nave 2
                draw_match_numbers2(); //desenha a vida das naves
                draw_tiro(addr, tiro_control); //desenha o tiro da nave 1
                draw_tiro2(addr, tiro2_control); //desenha o tiro da nave2
                test_colision_nave1(nave_x, nave_y, 1); //testa se o tiro da nave 2
colidiu com a nave 1
                test_colision_nave2(nave_x2, nave_y2, 2); //testa se o tiro da nave 1
colidiu com a nave 2

                if (!tst_bit(PINB, BOTAO_DOWN) && ((nave_y + tam_nave) < 62))
                { //testa para ver se o botao 2 mudou de estado
                    nave_y++; //se sim incrementa o valor de y da nave 1
                }
                if (!tst_bit(PINB, BOTAO_UP) && (nave_y > 2))
                { //testa para ver se o botão 1 mudou de estado
                    nave_y--; //se sim decrementa o valor do y da nave 1
                }
                if (!tst_bit(PINB, SELECT))
                { //testa para ver se o botão 3 mudou de estado
                    if (!tiro_control) //verifica se pode atirar
                    {
                        tiro_control = true; //se sim, altera o controle para true
                    }
                }
                if ((score_nave1 == 0) || (score_nave2 == 0))
                { //se a vida das naves chegar a 0, inicia o processo de fim de jogo
                    start_game2 = 0; //anula o estado do jogo
                    break; //sai do loop
                }

                SSD1306_UpdateScreen(addr); //atualiza a tela com os objetos
desenhados
            }
            if (start_game2 == 0) //testa se deve ativar a tela de fim de jogo
            {
                game_over_screen2(addr); //caso a nave 1 tenha ganho, apresenta na
tela, se nao, apresenta que foi a nave 2
                break; // finaliza o jogo e volta ao menu principal de jogos
            }
        }
    }
}
```

Para que a função de desenhar bitmaps funcionasse, foram necessárias algumas modificações em **ssd1306.c**. Foi criado a função `drawBitMap()`, utilizando como base a função `SSD1306_DrawPixel` foi possível desenhar qualquer imagem de até 32x32px, por meio de uma matriz de pixels que deve estar ordenada verticalmente, seu funcionamento é apresentado no código abaixo:

```
// desenha os bitmaps utilizados
```

```
void drawBitMap(uint8_t x, uint8_t y, const uint8_t *map, uint8_t linhas, uint8_t
colunas)
{

  uint8_t w = 0;       // determina o contador das colunas a serem lidas
  uint8_t k = 0;       // determina o contador das linhas a serem lidas
  uint8_t ajuste = 0;  // utilizado para atualizar o grupo de pixeis a ser ligado
  uint8_t ajuste2 = 0; // atualiza o index dos bytes a serem lidos
  while (w < colunas)  // percorre todas as colunas
  {

    while (k < linhas) // percorre todas as linhas
    {
      for (int8_t l = 7; l > -1; l--) //laco para testar os 8 bits passados
      {
        if (tst_bit(map[k + ajuste2], l)) //testa para ver se o pixel deve ligar ou
nao
        {
          // se sim, liga o pixel na posicao determinada pelo bit e pelo ajuste
          SSD1306_DrawPixel(x + l + ajuste, y + k);
          // y é atualizado a partir das linhas
          // o bitmap deve estar arranjado verticalmente
        }
      }
      k++; // atualiza o valor das linhas
    }
    k = 0;         // reseta o valor das linhas
    ajuste += 8;   // ajusta para a proxima coluna de bytes
    ajuste2 += 32; // ajusta o index do bitmap
    w++;           // atualiza o valor da coluna
  }
}
```

Referências

SANTOS, Andressa MS et al. RPG Arduino: uma proposta de gamificação para ensino de programação em microcontroladores. In: **Anais da XXI Escola Regional de Computação Bahia, Alagoas e Sergipe**. SBC, 2021. p. 161-170.

GitHub. 2022. *GitHub - Matiasus/SSD1306: C library for SSD1306 OLED Driver* . [online] Disponível em: https://github.com/Matiasus/SSD1306 [Accessado em: 18 June 2022].

Apêndice I

main.c

```
// frequencia de operacao de 16MHz
#include <avr/io.h>
#include <avr/pgmspace.h>
#include <util/delay.h>
#include <menu.h>
#include <ssd1306.h>
```

```
// Definicoes de macros
#define set_bit(Y, bit_x) (Y |= (1 << bit_x))  // ativa bit
#define clr_bit(Y, bit_x) (Y &= ~(1 << bit_x)) // limpa bit

uint8_t addr = SSD1306_ADDRESS;
int main()
{
    while (1)
    {
        menu(addr);
        _delay_ms(1000);
    }
}
```

---

ssd1306.h modificado, apontados por setas.

```
  uint8_t SSD1306_DrawCharAt(uint8_t, uint8_t, int);               //<-------

  void drawBitMap(uint8_t , uint8_t , const uint8_t* , uint8_t,uint8_t); //<----
```

---

ssd1306.c, modificações marcadas com uma seta.

```
uint8_t SSD1306_DrawCharAt(uint8_t x, uint8_t y, int a) //<-----------------
{
  char num = 'a';
  uint8_t i = 0;
  uint8_t page = 0;
  // uint8_t pixel = 0;

  // if out of range
  if ((x > MAX_X) && (y > MAX_Y))
  {
    // out of range
    return SSD1306_ERROR;
  }
  // find page (y / 8)
  page = y >> 3;
  // which pixel (y % 8)
  // pixel = 1 << (y - (page << 3));
  // update counter
  _counter = x + (page << 7);
  // save pixel

  if (a == 0)
  {
    num = '0';
  }
  else if (a == 1)
  {
    num = '1';
  }
  else if (a == 2)
  {
    num = '2';
  }
```

```
  else if (a == 3)
  {
    num = '3';
  }
  else if (a == 4)
  {
    num = '4';
  }
  else if (a == 5)
  {
    num = '5';
  }

  while (i < CHARS_COLS_LENGTH)
  {
    cacheMemLcd[_counter++] = pgm_read_byte(&FONTS[num - 32][i++]);
  }
  // success
  return SSD1306_SUCCESS;
}

// desenha os bitmaps utilizados
void drawBitMap(uint8_t x, uint8_t y, const uint8_t *map, uint8_t linhas, uint8_t
colunas)
{

  uint8_t w = 0; //determina o contador das colunas a serem lidas
  uint8_t k = 0; //determina o contador das linhas a serem lidas
  uint8_t ajuste = 0; // utilizado para atualizar o grupo de pixeis a ser ligado
  uint8_t ajuste2 = 0; //atualiza o index dos bytes a serem lidos
  while (w < colunas) //percorre todas as colunas
  {

    while (k < linhas) //percorre todas as linhas
    {
      for (int8_t l = 7; l > -1; l--) //laco para testar os 8 bits passados
      {
        if (tst_bit(map[k+ajuste2], l)) //testa para ver se o pixel deve ligar ou nao
        {
            //se sim, liga o pixel na posicao determinada pelo bit e pelo ajuste
          SSD1306_DrawPixel(x +l+ajuste , y + k);
            //y é atualizado a partir das linhas
            // o bitmap deve estar arranjado verticalmente
        }
      }
      k++;// atualiza o valor das linhas
    }
    k = 0;//reseta o valor das linhas
    ajuste+=8;// ajusta para a proxima coluna de bytes
    ajuste2+=32;//ajusta o index do bitmap
    w++;//atualiza o valor da coluna
  }
}
```

pong.h

```
#ifndef __PONG_H__
#define __PONG_H__

void pong_init(uint8_t);

#endif
```

pong.c

```
#include <avr/io.h> //definições do componente especificado
#include <avr/pgmspace.h>
#include <util/delay.h>
//#include <Arduino.h>
// include libraries
#include <stdio.h>
#include <ssd1306.h>
#include <stdlib.h>

//====================Definições====================================
#define set_bit(Y, bit_x) (Y |= (1 << bit_x))  // ativa o bit x da variável Y (coloca em 1)
#define clr_bit(Y, bit_x) (Y &= ~(1 << bit_x)) // limpa o bit x da variável Y (coloca em 0)
#define tst_bit(Y, bit_x) (Y & (1 << bit_x))   // testa o bit x da variável Y (retorna 0 ou 1)
//#define cpl_bit(Y, bit_x) (Y ^= (1 << bit_x))  // troca o estado do bit x da variável Y (complementa)
#define BOTAO_UP PB4
#define BOTAO_DOWN PB3
#define SELECT PB2
// define configuracoes som
#define som PB1               // pino OC1A para saida do sinal
#define d_inic 4              // valor inicial de duracao da nota musical
#define o_inic 5              // valor inicial da oitava
#define b 192                 /*o nr de batidas indica a velocidade da musica (alterar para mudar a velocidade), maior = mais rapido*/
#define t_min (7500 / b) * 10 // tempo minimo para formar o tempo base das notas musicais (1/32)
// define configuracoes jogo
#define true 1
#define false 0
#define nada -1
#define tam_plat 15 // tem q ser multiplo de 3

// variaveis para o pong
int8_t player1_x = 9;
int8_t player1_y = 26;
int8_t player2_x = 118;
int8_t player2_y = 26;
int8_t ball_x = 62;
int8_t ball_y = 30;
int8_t score_pl_1 = 0;
int8_t score_pl_2 = 0;
int8_t invert_ball_y = true;
int8_t invert_ball_x = true;
int8_t start_game = false;
```

```
int8_t ball_direction_preset = 1;
uint8_t dificuldade = 90;
// musica
const char musica[] PROGMEM = {"8f,8f"};


// functions for pong
void draw_Box(void);  //desenha bordas
void draw_player_1(void); //desenha jogador 1
void draw_player_2(void); //desenha jogador 2 e controla a I.A.
void drawBall(void);  //atualiza a posicao e desenha a bola
void update_ball_x(void); //atualiza a posicao da bola baseado no invert_ball_x
void update_ball_y(void); //atualiza a posicao da bola baseado no invert_ball_y
void invert_ball_direction(void); // determina se deve inverter as variaveis
Invert_ball
void draw_match_numbers(void); // desenha na tela o placar
void game_over_screen(uint8_t); //desenha a tela de fim de jogo e redefine as
variaveis do jogo
void init_screen(uint8_t); // apresenta tela de inicio do jogo
void test_colision(uint8_t, uint8_t, int8_t); //testa se a bola colidiu com alguma
jogador
void toca_musica(const char *); // toca o som da colisao
void pong_init(uint8_t); // funcao principal


void pong_init(uint8_t address)
{

  int addr = address; // armazena o endereco de comandos da tela
  DDRB = 0b00000000;  // portas 2,1 e 0 como entrada;
  PORTB = 0b11111111; // habilita pullup para entradas
  // inicia pong
  SSD1306_ClearScreen();
  // INICIA SETUP DO JOGO
  while (1)
  {
    init_screen(addr);// desenha a tela de inicio do jogo

    if (!tst_bit(PINB, BOTAO_UP)) // volta para o menu principal
    {
      break;
    }
    if (!tst_bit(PINB, SELECT)) // inicia o jogo
    {
      start_game = true;
    }

    if (start_game) // o jogo
    {
      while (1)
      {
        SSD1306_ClearScreen(); //limpa a tela
        drawBox(); //desenha as bordas
        draw_match_numbers(); //desenha o placar
        draw_player_1(); //desenha o jogador 1
        draw_player_2(); //desenha o jogador 2
        drawBall(); // desenha a bola
```

```
        if (!tst_bit(PINB, BOTAO_DOWN) && ((player1_y + tam_plat) < 62))
        {//testa para ver se o botao 2 mudou de estado
          player1_y++; //se sim incrementa o valor de y do jogador 1
        }
        if (!tst_bit(PINB, BOTAO_UP) && (player1_y > 1))
        {//testa para ver se o botão 1 mudou de estado
          player1_y--;//se sim decrementa o valor do y do jogador 1
        }
        if ((score_pl_1 == 3) || (score_pl_2 == 3))
        {//se o placar de qualquer um dos jogadores chegar a 3, inicia o processo de
fim de jogo
          start_game = 0; //anula o estado do jogo
          break; //sai do loop
        }
        SSD1306_UpdateScreen(addr);//atualiza a tela com os objetos desenhados
      }
      if (start_game == 0)//testa se deve ativar a tela de fim de jogo
      {
        game_over_screen(addr);//caso o jogador 1 tenha ganho, apresenta na tela, se
nao, apresenta que foi o jogador numero 2.
        break;// finaliza o jogo e volta ao menu principal de jogos
      }
    }
  }
}

// desenha arena
void draw_Box()
{
  SSD1306_DrawLine(0, MAX_X, 0, 0);
  SSD1306_DrawLine(0, MAX_X, 63, 63);
  SSD1306_DrawLine(0, 0, 0, MAX_Y);
  SSD1306_DrawLine(127, 127, 0, MAX_Y);
}

// desenha o player 1
void draw_player_1()
{
  SSD1306_DrawLine(player1_x, player1_x, (player1_y), (player1_y + tam_plat));
  // SSD1306_UpdateScreen(addr);
}

// desenha a barra do jogador numero 2
void draw_player_2()
{
  if (ball_x > dificuldade) // controla I.A. do jogador numero 2,
  {                         //começa a se mover
                               //quando a bola_x chega em 74.

    if ((ball_y < player2_y) && (player2_y > 1))  //se a posiçao y da bola for
    {                                             //inferior ao y do jogador 2,
                                                  //eleva a barra
      player2_y--;
    }
    if ((ball_y > player2_y) && ((player2_y + tam_plat) < 62)) //se a posicao y
    {                                                          //da bola for superior
       player2_y++;                                          //a do jogador 2, rebaixa
                                                          //a barra.
```

```
    }
  }

  SSD1306_DrawLine(player2_x, player2_x, (player2_y), (player2_y + tam_plat));
}

// desenha a bola
void drawBall()
{
  invert_ball_direction();
  SSD1306_DrawPixel(ball_x, ball_y);
}

// atualiza posicao x da bola caso bata em algum dos players
void update_ball_x()
{
  if (!invert_ball_x)
  {
    ball_x--;
  }
  else if (invert_ball_x)
  {
    ball_x++;
  }
}

// atualiza a direçao da bola em y
void update_ball_y()
{
  if (!invert_ball_y)
  {
    ball_y--;
  }
  else if (invert_ball_y)
  {
    ball_y++;
  }
  else if (invert_ball_y == nada)
  {
    ball_y = ball_y;
  }
}

// inverte direcao da bola
void invert_ball_direction()
{
  // verifica se a bola bateu em alguma das paredes horizontais
  if ((ball_y == 1) || (ball_y == 62))
  {
    // se sim, inverte sua direcao em y
    invert_ball_y = !invert_ball_y;
  }
  // verifica se a bola chegou na coluna dos players
  test_colision(player1_x, player1_y,1);
  test_colision(player2_x, player2_y,-1);
  // testa se a bola ultrapassou os players, ou seja, nao houve colisao
  if ((ball_x == 1) || (ball_x == 126))
  {
```

```
    if (ball_x == 1) // se ultrapassar o jogador 1
    {
      score_pl_2++; //adiciona ponto para o jogador 2
    }
    else if (ball_x == 126) //se ultrapassar o jogador 2
    {
      score_pl_1++;  //adiciona ponto ao jogador 1
    }
    // ao fazer gol, reseta a posicao da bola para o centro da tela
    ball_x = MAX_X / 2;
    ball_y = MAX_Y / 2;

 switch (ball_direction_preset) // controla a pseudo - aleatoriedade da bola
    {
    case 1: //nao inverte a direcao da bola
      invert_ball_x = invert_ball_x;
      invert_ball_y = invert_ball_y;
      break;
    case 2: //inverte apenas a direcao de x
      invert_ball_x = !invert_ball_x;
      invert_ball_y = invert_ball_y;
      break;
    case 3: // inverte a direcao de y apenas
      invert_ball_x = invert_ball_x;
      invert_ball_y = !invert_ball_y;
      break;
    case 4: // inverte totalente a direcao
      invert_ball_x = !invert_ball_x;
      invert_ball_y = !invert_ball_y;
      break;
    }
    ball_direction_preset++; // muda o preset da bola
  }
  update_ball_x(); //atualiza se deve incrementar ou decrementar, baseado nas mudanças
feitas nos testes de colisao.
  update_ball_y();
}

// desenha placar
void draw_match_numbers(void)
{
  SSD1306_DrawCharAt(42, 10, (char)score_pl_1);
  SSD1306_DrawCharAt(84, 10, (char)score_pl_2);
}

// desenha tela de game over
void game_over_screen(uint8_t address)
{
  int addr = address;
  SSD1306_ClearScreen();
  draw_Box();
  if (score_pl_1 == 3)
  {
    SSD1306_SetPosition(40, 1);
    SSD1306_DrawString("GAME OVER");
    SSD1306_DrawLine(7, 120, 18, 18);
    SSD1306_SetPosition(6, 4);
    SSD1306_DrawString("PLAYER 1 WINS");
```

```
  }
  else if (score_pl_2 == 3)
  {
    SSD1306_SetPosition(40, 1);
    SSD1306_DrawString("GAME OVER");
    SSD1306_DrawLine(7, 120, 18, 18);
    SSD1306_SetPosition(6, 4);
    SSD1306_DrawString("PLAYER 2 WINS");
  }
  SSD1306_UpdateScreen(addr);
  _delay_ms(2000);
  SSD1306_ClearScreen();
  // redefine variaveis
  player1_x = 9;
  player1_y = 26;
  player2_x = 118;
  player2_y = 26;
  ball_x = 62;
  ball_y = 30;
  score_pl_1 = 0;
  score_pl_2 = 0;
  invert_ball_y = true;
  invert_ball_x = true;
  start_game = false;
}

// desenha tela inicial
void init_screen(uint8_t address)
{
  int addr = address;
  draw_Box();
  SSD1306_SetPosition(40, 1);
  SSD1306_DrawString("PONG GAME");
  SSD1306_DrawLine(7, 120, 18, 18);
  SSD1306_SetPosition(4, 4);
  SSD1306_DrawString("PRESS 3 TO BEGIN");
  SSD1306_SetPosition(4, 6);
  SSD1306_DrawString("PRESS 1 TO EXIT");
  SSD1306_UpdateScreen(addr);
}

void test_colision(uint8_t player_x, uint8_t player_y, int8_t ajuste)
{
  int player_y_end = (tam_plat + player_y); // calcula o valor de y final para o
jogador
  if (player_x + ajuste == ball_x) // define o pixel o qual o teste começa
  {
    uint8_t fragment_of_paddle = tam_plat / 3; // calcula o tamanho das seções do
jogador, para que a physica da bola seja diferente entre a parte superior, central e
inferior
    uint8_t lim_part_1 = player_y_end - (fragment_of_paddle * 2);//tamanho de 2/3 da
barra
    uint8_t lim_part_2 = player_y_end - (fragment_of_paddle); //tamanho de 1/3 da
barra
    if ((ball_y >= player_y) && (ball_y <= lim_part_1)) // testa para ver se a bola
colidiu na parte superior do jogador
    {
        //se sim, inverte x e diminui y, a bola sobe na diagonal superior
```

```
      invert_ball_x = !invert_ball_x;
      invert_ball_y = false;
      toca_musica(musica); //reproduz som de colisao
    }
    else if ((ball_y >= lim_part_1) && (ball_y <= lim_part_2)) //testa se a bola
colidiu na parte central do jogador
    {//se sim, inverte x e anula y, a bola deve seguir reta neste caso
      invert_ball_x = !invert_ball_x;
      invert_ball_y = nada;
      toca_musica(musica);
    }
    else if ((ball_y >= lim_part_2) && (ball_y <= player_y_end))// testa se a bola
colidiu na parte inferior do jogador
    { //se sim, inverte x e aumenta y, a bola sobe na diagonal inferior
      invert_ball_x = !invert_ball_x;
      invert_ball_y = true;
      toca_musica(musica);
    }

  } /*
    if (ball_y >= player_y && ball_y <= player_y_end)
    {
      invert_ball_x = !invert_ball_x;
       toca_musica(musica);
    }*/
}

void toca_musica(const char *musica)
{
  // configura e define a musica de inicio
  unsigned int k;
  unsigned char d, o, j, n, i = 0;
  DDRB |= (1 << som); // habilita a saida de som

  OCR1A = 18181;

  // TCCR1A = 1<<COM1A0;
  // TC1 modo CTC comparacao com OCR1A, prescaler=1
  TCCR1B = (1 << WGM12) | (1 << CS10);
  //-----------------------------------------------------------------------------
  // LEITURA E EXECUCAO DO ARQUIVO RTTTL
  //-----------------------------------------------------------------------------
  do
  {
    o = o_inic; // carrega o valor default para a oitava
    if ((pgm_read_byte(&musica[i]) == '3') && (pgm_read_byte(&musica[i + 1]) == '2'))
    {
      d = 32;
      i += 2;
    }
    else if ((pgm_read_byte(&musica[i]) == '1') && (pgm_read_byte(&musica[i + 1]) ==
'6'))
    {
      d = 16;
      i += 2;
    }
    else if (pgm_read_byte(&musica[i]) == '8')
    {
```

```
      d = 8;
      i++;
    }
    else if (pgm_read_byte(&musica[i]) == '4')
    {
      d = 4;
      i++;
    }
    else if (pgm_read_byte(&musica[i]) == '2')
    {
      d = 2;
      i++;
    }
    else if (pgm_read_byte(&musica[i]) == '1')
    {
      d = 1;
      i++;
    }
    else
      d = d_inic; // carrega o valor dafault para a duracao
    if (pgm_read_byte(&musica[i + 1]) == '#')
    {
      switch (pgm_read_byte(&musica[i]))
      { // carrega a oitava # default (4a)
      case 'a':
        OCR1A = 17159;
        break; // A# - La#
      case 'c':
        OCR1A = 14429;
        break; // C# - Do#
      case 'd':
        OCR1A = 12852;
        break; // D# - Re#
      case 'f':
        OCR1A = 10807;
        break; // F# - Fa#
      case 'g':
        OCR1A = 9627;
        break; // G# - Sol#
      }
      i += 2;
    }
    else
    {
      switch (pgm_read_byte(&musica[i]))
      { // carrega a oitava default (4a)
      case 'a':
        OCR1A = 18181;
        break; // A - La
      case 'b':
        OCR1A = 16197;
        break; // B - Si
      case 'c':
        OCR1A = 15287;
        break; // C - Do
      case 'd':
        OCR1A = 13618;
        break; // D - Re
```

```
      case 'e':
        OCR1A = 12131;
        break; // E - Mi
      case 'f':
        OCR1A = 11449;
        break; // F - Fa
      case 'g':
        OCR1A = 10199;
        break; // G - Sol
      case 'p':
        clr_bit(TCCR1A, COM1A0);
        break; // p = pausa
      }
      i++;
    }
    n = 32 / d; // tempo de duracao de cada nota musical

    if (pgm_read_byte(&musica[i]) == '.')
    {
      n = n + (n / 2); // duracao 50% >
      i++;
    }
    if (pgm_read_byte(&musica[i]) == '4')
    {
      o = 4;
      i++;
    }
    else if (pgm_read_byte(&musica[i]) == '5')
    {
      o = 5;
      i++;
    }
    else if (pgm_read_byte(&musica[i]) == '6')
    {
      o = 6;
      i++;
    }
    else if (pgm_read_byte(&musica[i]) == '7')
    {
      o = 7;
      i++;
    }
    if (pgm_read_byte(&musica[i]) == '.')
    {
      n = n + (n / 2); // duracao 50% >
      i++;
    }
    switch (o)
    { // troca a oitava se nao for a default (o = 4)
    case 5:
      OCR1A = OCR1A >> 1;
      break; // divide por 2
    case 6:
      OCR1A = OCR1A >> 2;
      break; // divide por 4
    case 7:
      OCR1A = OCR1A >> 4;
      break; // divide por 8
```

```
    }
    //-------------------------------------------------------------------------
    for (j = 0; j < n; j++)
    { // nr de repeticoes para a nota 1/32
      for (k = t_min; k != 0; k--)
        _delay_us(100);
    }
    //-------------------------------------------------------------------------
    set_bit(TCCR1A, COM1A0);                    // habilita o som
  } while (pgm_read_byte(&musica[i++]) == ','); // leitura ate o final da musica
  TCCR1A = 0;                                   // desabilita o som e o TC1
  TCCR1B = 0;
}
```

space.h

```
#ifndef __SPACE_H__
#define __SPACE_H__

void space_init(uint8_t);

#endif
```

space.c

```
#include <avr/io.h> //definições do componente especificado
#include <avr/pgmspace.h>
#include <util/delay.h>
//#include <Arduino.h>
// include libraries
#include <stdio.h>
#include <ssd1306.h>
#include <stdlib.h>

//=====================Definições===================================
#define set_bit(Y, bit_x) (Y |= (1 << bit_x))  // ativa o bit x da variável Y (coloca
em 1)
#define clr_bit(Y, bit_x) (Y &= ~(1 << bit_x)) // limpa o bit x da variável Y (coloca
em 0)
#define tst_bit(Y, bit_x) (Y & (1 << bit_x))   // testa o bit x da variável Y (retorna
0 ou 1)
//#define cpl_bit(Y, bit_x) (Y ^= (1 << bit_x)) // troca o estado do bit x da variável
Y (complementa)
#define BOTAO_UP PB4
#define BOTAO_DOWN PB3
#define SELECT PB2
// define configuracoes jogo
#define true 1
#define false 0
#define nada -1
#define TAM_TIRO 5  // Tamanho do tiro da nave 1
#define TAM_TIRO2 7 // Tamanho do tiro da nave 2

// define configuracoes som
#define som PB1               // pino OC1A para sa◆da do sinal
```

```
#define d_inic 4              // valor inicial de dura��o da nota musical
#define o_inic 5              // valor inicial da oitava
#define b 192                 /*o nr de batidas indica a velocidade da musica (alterar
para mudar a velocidade), maior = mais r�pido*/
#define t_min (7500 / b) * 10 // tempo m�nimo para formar o tempo base das notas
musicais (1/32)

// variaveis para o space
int start_game2 = false;

int nave_x = 9;
int nave_y = 26;
int tam_nave = 11;

int nave_x2 = 95;
int nave_y2 = 16;
int tam_nave2 = 32;

int tiro_y = 0;
int tiro_x = 0;

int tiro2_y = 0;
int tiro2_x = 0;

uint8_t tiro_control = false;
uint8_t tiro2_control = false;

uint8_t nave2_control = false;

int score_nave1 = 3;
int score_nave2 = 5;

const char musica2[] PROGMEM = {"8f,8f"}; // musica

// Bitmaps
const uint8_t st_bitmap_player1[] =
    {
        0x06, 0x1e, 0x06, 0x07, 0x1f, 0x7e, 0x1f, 0x07, 0x06, 0x1e, 0x06};

const uint8_t st_bitmap_player2[] =
    {
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x80, 0x80,
        0x80, 0xc0, 0xc0, 0x80, 0x80, 0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xe0, 0xf8, 0xfc, 0xfe,
0xff, 0xff, 0xe3,
        0xc9, 0x1c, 0x9d, 0xc9, 0xe3, 0xff, 0xff, 0xfe, 0xfc, 0xf8, 0xc0, 0x00, 0x00,
0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x07, 0x01, 0x01, 0x7f,
0x07, 0xff, 0x01,
        0x01, 0xfe, 0x07, 0xff, 0x01, 0xff, 0xff, 0x07, 0x3f, 0x1f, 0x03, 0x00, 0x00,
0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x01, 0x00,
        0x00, 0x03, 0x00, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00};
```

```
const uint8_t st_bitmap_tiro_player2[] =
    {
        /* 00111000 */ 0x038,
        /* 01111100 */ 0x07c,
        /* 11111100 */ 0x0fc,
        /* 11111110 */ 0x0fe,
        /* 11111110 */ 0x0fe,
        /* 01111110 */ 0x07e,
        /* 01111000 */ 0x078};

// functions for space
void drawBox2(void);                                    // desenha bordas
void draw_nave(int);                                    // desenha nave 1
void draw_nave2(int);                                   // desenha nave 2 e controla a
I.A
void init_screen2(uint8_t);                             // apresenta a tela inicial do
jogo
void space_init(uint8_t);                               // função principal
void draw_tiro(uint8_t, uint8_t);                       // desenha o tiro da nave 1 e
atualiza a posicao e condição para atirar
void draw_tiro2(uint8_t, uint8_t);                      // desenha o tiro da nave 2 e
atualiza a posicao e condição para atirar
void test_colision_nave1(uint8_t, uint8_t, uint8_t); // testa se o tiro da nave 1
colidiu com a nave 2
void test_colision_nave2(uint8_t, uint8_t, uint8_t); // testa se o tiro da nave 2
colidiu com a nave 1
void toca_musica2(const char *);                        // toca o som da colisão
void draw_match_numbers2(void);                         // desenha na tela a vida das
naves
void game_over_screen2(uint8_t);                        // desenha a tela de fim de jogo
e redefine as variaveis do jogo

void space_init(uint8_t address)
{

    int addr = address; // armazena o endereco de comandos da tela
    DDRB = 0b00000000;  // portas 2,1 e 0 como entrada;
    PORTB = 0b11111111; // habilita pullup para entradas
    // inicia space
    SSD1306_ClearScreen();
    // INICIA SETUP DO JOGO
    while (1)
    {
        init_screen2(addr);

        if (!tst_bit(PINB, BOTAO_UP)) // volta para menu principal
        {
            break;
        }
        if (!tst_bit(PINB, SELECT)) // inicia o jogo
        {
            start_game2 = true;
        }

        if (start_game2) // o jogo
        {
            while (1)
            {
```

```
                SSD1306_ClearScreen(); //limpa a tela
                drawBox2(); //desenha as bordas
                draw_nave(addr); //desenha a nave 1
                draw_nave2(addr); //desenha a nave 2
                draw_match_numbers2(); //desenha a vida das naves
                draw_tiro(addr, tiro_control); //desenha o tiro da nave 1
                draw_tiro2(addr, tiro2_control); //desenha o tiro da nave2
                test_colision_nave1(nave_x, nave_y, 1); //testa se o tiro da nave 2
colidiu com a nave 1
                test_colision_nave2(nave_x2, nave_y2, 2); //testa se o tiro da nave 1
colidiu com a nave 2

                if (!tst_bit(PINB, BOTAO_DOWN) && ((nave_y + tam_nave) < 62))
                { //testa para ver se o botao 2 mudou de estado
                    nave_y++; //se sim incrementa o valor de y da nave 1
                }
                if (!tst_bit(PINB, BOTAO_UP) && (nave_y > 2))
                { //testa para ver se o botão 1 mudou de estado
                    nave_y--; //se sim decrementa o valor do y da nave 1
                }
                if (!tst_bit(PINB, SELECT))
                { //testa para ver se o botão 3 mudou de estado
                    if (!tiro_control) //verifica se pode atirar
                    {
                        tiro_control = true; //se sim, altera o controle para true
                    }
                }
                if ((score_nave1 == 0) || (score_nave2 == 0))
                { //se a vida das naves chegar a 0, inicia o processo de fim de jogo
                    start_game2 = 0; //anula o estado do jogo
                    break; //sai do loop
                }

                SSD1306_UpdateScreen(addr); //atualiza a tela com os objetos
desenhados
            }
            if (start_game2 == 0) //testa se deve ativar a tela de fim de jogo
            {
                game_over_screen2(addr); //caso a nave 1 tenha ganho, apresenta na
tela, se nao, apresenta que foi a nave 2
                break; // finaliza o jogo e volta ao menu principal de jogos
            }
        }
    }
}

// desenha arena
void drawBox2()
{
    SSD1306_DrawLine(0, MAX_X, 0, 0);
    SSD1306_DrawLine(0, MAX_X, 63, 63);
    // SSD1306_DrawLine(0, 0, 0, MAX_Y);
    // SSD1306_DrawLine(127, 127, 0, MAX_Y);
}

// desenha a nave 1
void draw_nave(int addr)
{
```

```
    drawBitMap(nave_x, nave_y, st_bitmap_player1, 11, 1);
}

// desenha a nave 2
void draw_nave2(int addr)
{

    if (nave_y2 + 5 > 0 && nave2_control == false) // Se a posição y da nave for
superior  0, eleva a nave
    {                                           //É adicionado +5 para a nave parar
5 pixels antes da barra superior
        nave_y2--;                              // nave2_control para controle de
prioridade da nave seguir pra uma direção
    }                                           // false: para nave subir e true:
para nave descer.
    if (nave_y2 + 5 == 1)
    {
        nave2_control = true; // a nave chegou ao limite superior, então alterou o
controle para poder descer
    }
    if ((nave_y2 + tam_nave2 - 3) < 63 && nave2_control == true) // Se a posição y da
nave for inferior a 63, rebaixa a nave
    {                                                       //É adicionado -3
para a nave parar 3 pixels antes da barra inferior
        nave_y2++;                                          //+ o tam_nave2 para
o desenho não ultrapassar o limite
    }                                                       // nave2_control para
controle de prioridade da nave seguir pra uma direção
    if ((nave_y2 + tam_nave2 - 3) == 62)
    {
        nave2_control = false; // a nave chegou ao limite inferior, então alterou o
controle para poder subir
    }

    drawBitMap(nave_x2, nave_y2, st_bitmap_player2, tam_nave2, 4); // chama o metódo
para desenhar o bitmap (classe: SSD1306)
}

// desenha o tiro 1
void draw_tiro(uint8_t addr, uint8_t controle)
{
    // controle é iniciado como false
    if (!controle)
    {
        tiro_y = (nave_y + (tam_nave / 2)); // calcula para que o tiro seja disparado
do meio da nave
        tiro_x = (nave_x + 8);              // calcula para que o tiro seja disparado
da frente da nave
    }
    // Quando o botão é precinado o controle é alterado para true
    if (controle)
    {
        if (tiro_x + TAM_TIRO == 110) // Verifica se o tiro chegou no limite da tela
        {
            tiro_control = false; // Caso verdadeiro, apaga o desenho.
            _delay_ms(250);
        }
```

```
        SSD1306_DrawLine(tiro_x, (tiro_x + TAM_TIRO), tiro_y, tiro_y); // chama o
metódo para desenhar a linha do tiro (classe: SSD1306)
        tiro_x++;                                                      // incrementa
em x para movimentar o desenho
    }
}

// desenha o tiro 2
void draw_tiro2(uint8_t addr, uint8_t controle2)
{
    // controle é iniciado como false
    if (!controle2)
    {
        tiro2_y = (nave_y2 + (tam_nave2 / 2)); // calcula para que o tiro seja
disparado do meio da nave
        tiro2_x = (nave_x2 - 2);               // calcula para que o tiro seja
disparado da frente da nave
    }
    // Nave 2 atira no centro da nave 1
    if ((nave_y - 14) == nave_y2)
    {
        tiro2_control = true; // controle é alterado para true para o mesmo disparar
    }
    // controle é alterado para true quando a nave 1 estiver no centro da nave 2
    if (controle2)
    {
        if (tiro2_x == 7) // Verifica se o tiro chegou no limite da tela
        {
            tiro2_control = false; // Caso verdadeiro, apaga o desenho.
        }

        drawBitMap(tiro2_x, tiro2_y, st_bitmap_tiro_player2, TAM_TIRO2, 1); // chama o
metódo para desenhar o bitmap (classe: SSD1306)
        tiro2_x--;                                                          //
decrementa em x para movimentar o desenho da direita para esquerda
    }
}

// desenha na tela o placar
void draw_match_numbers2(void)
{
    SSD1306_DrawCharAt(42, 10, (char)score_nave1);
    SSD1306_DrawCharAt(84, 10, (char)score_nave2);
}

// desenha a tela de fim de jogo e redefine as variaveis do jogo
void game_over_screen2(uint8_t address)
{
    int addr = address;
    SSD1306_ClearScreen();
    drawBox2();
    if (score_nave1 == 0)
    {
        SSD1306_SetPosition(40, 1);
        SSD1306_DrawString("GAME OVER");
        SSD1306_DrawLine(7, 120, 18, 18);
        SSD1306_SetPosition(6, 4);
        SSD1306_DrawString("DEATH STAR WINS");
```

```
    }
    else if (score_nave2 == 0)
    {
        SSD1306_SetPosition(40, 1);
        SSD1306_DrawString("GAME OVER");
        SSD1306_DrawLine(7, 120, 18, 18);
        SSD1306_SetPosition(6, 4);
        SSD1306_DrawString("PLAYER WINS");
    }
    SSD1306_UpdateScreen(addr);
    _delay_ms(2000);
    SSD1306_ClearScreen();
    // redefine variaveis

    nave_x = 9;
    nave_y = 26;
    tam_nave = 11;

    tiro_y = 0;
    tiro_x = 0;

    tiro2_y = 0;
    tiro2_x = 0;

    tiro_control = false;
    tiro2_control = false;

    nave2_control = false;

    nave_x2 = 95;
    nave_y2 = 16;
    tam_nave2 = 32;

    score_nave1 = 3;
    score_nave2 = 5;

    start_game2 = false;
}

// apresenta a tela inicial do jogo
void init_screen2(uint8_t address)
{
    int addr = address;
    drawBox2();
    SSD1306_SetPosition(40, 1);
    SSD1306_DrawString("SPACE GAME");
    SSD1306_DrawLine(7, 120, 18, 18);
    SSD1306_SetPosition(4, 4);
    SSD1306_DrawString("PRESS 3 TO BEGIN");
    SSD1306_SetPosition(4, 6);
    SSD1306_DrawString("PRESS 1 TO EXIT");
    SSD1306_UpdateScreen(addr);
}

// testa se o tiro da nave 2 colidiu com a nave 1
void test_colision_nave1(uint8_t player_x, uint8_t player_y, uint8_t ajuste)
{
```

```
    int player_y_end = (tam_nave + player_y); // calcula o valor de y final para a
nave
    if (player_x + ajuste == tiro2_x)         // define o pixel o qual o teste começa
    {
        if (tiro2_y >= player_y && tiro2_y <= player_y_end) // testa para ver se o
tiro colidiu na nave 1
        {
            toca_musica2(musica2); // reproduz som de colisao
            score_nave1--;         // se, sim decrementa a vida da nave 1
        }
    }
}

// testa se o tiro da nave 1 colidiu com a nave 2
void test_colision_nave2(uint8_t player_x, uint8_t player_y, uint8_t ajuste)
{
    int player_y_end = (tam_nave2 + player_y - 2); // calcula o valor de y final para
a nave
    if (player_x + ajuste == tiro_x)               // define o pixel o qual o teste
começa
    {
        if (tiro_y >= player_y && tiro_y <= player_y_end) // testa para ver se o tiro
colidiu na nave 2
        {
            toca_musica2(musica2); // reproduz som de colisao
            score_nave2--;         // se, sim decrementa a vida ds nave 2
        }
    }
}

// toca o som da colisão
void toca_musica2(const char *musica2)
{
    // configura e define a musica de inicio
    unsigned int k;
    unsigned char d, o, j, n, i = 0;
    DDRB |= (1 << som); // habilita a sa�da de som

    OCR1A = 18181;

    // TCCR1A = 1<<COM1A0;
    // TC1 modo CTC compara��o com OCR1A, prescaler=1
    TCCR1B = (1 << WGM12) | (1 << CS10);
    //-------------------------------------------------------------------------
    // LEITURA E EXECU��O DO ARQUIVO RTTTL
    //-------------------------------------------------------------------------
    do
    {
        o = o_inic; // carrega o valor default para a oitava
        if ((pgm_read_byte(&musica2[i]) == '3') && (pgm_read_byte(&musica2[i + 1]) ==
'2'))
        {
            d = 32;
            i += 2;
        }
        else if ((pgm_read_byte(&musica2[i]) == '1') && (pgm_read_byte(&musica2[i +
1]) == '6'))
        {
```

```
            d = 16;
            i += 2;
        }
        else if (pgm_read_byte(&musica2[i]) == '8')
        {
            d = 8;
            i++;
        }
        else if (pgm_read_byte(&musica2[i]) == '4')
        {
            d = 4;
            i++;
        }
        else if (pgm_read_byte(&musica2[i]) == '2')
        {
            d = 2;
            i++;
        }
        else if (pgm_read_byte(&musica2[i]) == '1')
        {
            d = 1;
            i++;
        }
        else
            d = d_inic; // carrega o valor dafault para a dura��o
        if (pgm_read_byte(&musica2[i + 1]) == '#')
        {
            switch (pgm_read_byte(&musica2[i]))
            { // carrega a oitava # default (4a)
            case 'a':
                OCR1A = 17159;
                break; // A# - L�#
            case 'c':
                OCR1A = 14429;
                break; // C# - D�#
            case 'd':
                OCR1A = 12852;
                break; // D# - R�#
            case 'f':
                OCR1A = 10807;
                break; // F# - F�#
            case 'g':
                OCR1A = 9627;
                break; // G# - S�l#
            }
            i += 2;
        }
        else
        {
            switch (pgm_read_byte(&musica2[i]))
            { // carrega a oitava default (4a)
            case 'a':
                OCR1A = 18181;
                break; // A - L�
            case 'b':
                OCR1A = 16197;
                break; // B - Si
            case 'c':
```

```
                OCR1A = 15287;
                break; // C - D♯
            case 'd':
                OCR1A = 13618;
                break; // D - R♯
            case 'e':
                OCR1A = 12131;
                break; // E - Mi
            case 'f':
                OCR1A = 11449;
                break; // F - F♯
            case 'g':
                OCR1A = 10199;
                break; // G - Sol
            case 'p':
                clr_bit(TCCR1A, COM1A0);
                break; // p = pausa
            }
            i++;
        }
        n = 32 / d; // tempo de dura♯♯o de cada nota musical

        if (pgm_read_byte(&musica2[i]) == '.')
        {
            n = n + (n / 2); // dura♯♯o 50% >
            i++;
        }
        if (pgm_read_byte(&musica2[i]) == '4')
        {
            o = 4;
            i++;
        }
        else if (pgm_read_byte(&musica2[i]) == '5')
        {
            o = 5;
            i++;
        }
        else if (pgm_read_byte(&musica2[i]) == '6')
        {
            o = 6;
            i++;
        }
        else if (pgm_read_byte(&musica2[i]) == '7')
        {
            o = 7;
            i++;
        }
        if (pgm_read_byte(&musica2[i]) == '.')
        {
            n = n + (n / 2); // dura♯♯o 50% >
            i++;
        }
        switch (o)
        { // troca a oitava se n♯o for a default (o = 4)
        case 5:
            OCR1A = OCR1A >> 1;
            break; // divide por 2
        case 6:
```

```
            OCR1A = OCR1A >> 2;
            break; // divide por 4
        case 7:
            OCR1A = OCR1A >> 4;
            break; // divide por 8
        }
        //-----------------------------------------------------------------------
        for (j = 0; j < n; j++)
        { // nr de repeti��es para a nota 1/32
            for (k = t_min; k != 0; k--)
                _delay_us(100);
        }
        //-----------------------------------------------------------------------
        set_bit(TCCR1A, COM1A0);                    // habilita o som
    } while (pgm_read_byte(&musica2[i++]) == ','); // leitura at� o final da m�sica
    TCCR1A = 0;                                    // desabilita o som e o TC1
    TCCR1B = 0;
}
```