

MAZE RUNNER



1181153 - RODRIGO BRANQUINHO

1181186 - JOSÉ SOARES

Sumário

- Introdução
- Acelerómetro
- SPI (*Serial Peripheral Interface*)
- Configurações (GPIO, Timer...)
- Lógica do Jogo
- Demonstração



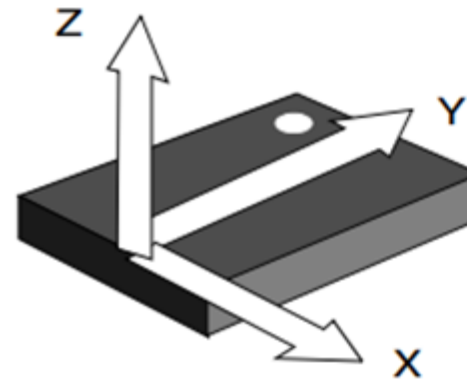
Introdução

- Objetivo: interpretar dados obtidos por um sensor no *Cortex M3*
- Escolha: Acelerómetro LIS3LV02DL (SPI)
- Ideia: Jogo do Labirinto (*Maze Runner*)
 - Visível no ecrã LCD
 - Jogador controlado pelo acelerómetro
 - Envio de dados para USART

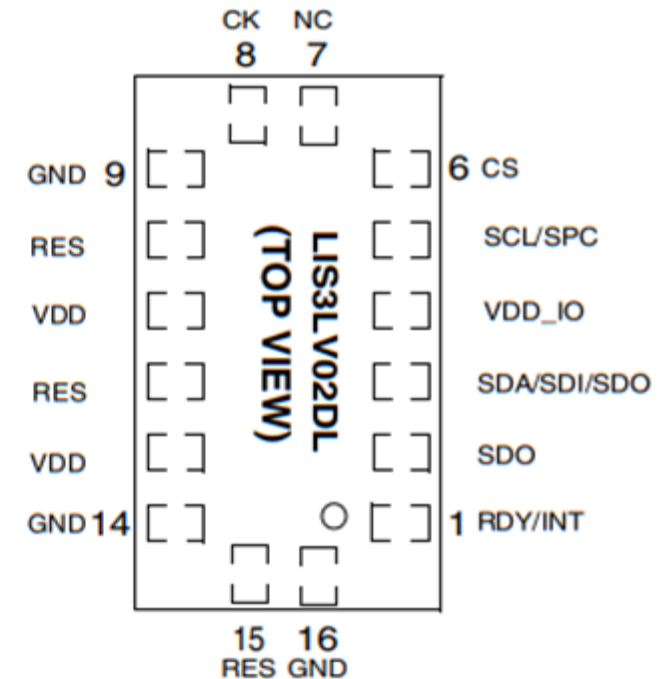


Acelerómetro – LIS3LV02DL

- Acelerómetro digital de 3 eixos
- Aplicações:
 - Detecção de queda livre
 - Sistemas anti-roubo
 - Jogos



EIXOS DAS
ACELERAÇÕES
DETETADAS PELO
ACELERÓMETRO



SPI (*Serial Peripheral Interface*)

- Comunicação síncrona
- Possui 4 ligações:
 - SCK (*Serial Clock*)
 - MOSI (*Master Output Slave Input*)
 - MISO (*Master Input Slave Output*)
 - CS/SS (*Chip Select/Slave Select*)



Configurações – GPIO

- Entradas:
 - Botão *SW5 (Pull-up)*
- Saídas:
 - LEDs (*Push-pull*)
 - TX USART (*Alternate Function*)

```
// SWITCHES: PA1 - SW5
RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_1;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_10MHz;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPU;
GPIO_Init(GPIOA, &GPIO_InitStructure);

// LEDs: PB0 PB1 PB2
RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE);
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0 | GPIO_Pin_1 | GPIO_Pin_2;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
GPIO_Init(GPIOB, &GPIO_InitStructure);

// USART2: PA2 - TX
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_2;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
GPIO_Init(GPIOA, &GPIO_InitStructure);
```

Configurações – GPIO (SPI)

- Entradas:
 - MISO (*In-floating*)
- Saídas:
 - RDY / SCK / MOSI (*Alternate Function*)
 - CS/SS (*Push-pull*)

```
// SPI: PB5 - RDY / PB13 - SCL (SPI2 SCK) / PB15 - SDA (SPI2 MOSI)
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_5 | GPIO_Pin_13 | GPIO_Pin_15;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
GPIO_Init(GPIOB, &GPIO_InitStructure);
// SPI: PB14 - SDO (SPI2 MISO)
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_14;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
GPIO_Init(GPIOB, &GPIO_InitStructure);
// SPI: PD2 - CS/SS
RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOD, ENABLE);
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_2;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
GPIO_Init(GPIOD, &GPIO_InitStructure);
```

Configurações – Relógio (RCC)

- Modo: HSE PLL
- Frequência: 72 MHz

```
void RCC_Config_HSE_PLL_Max() {  
    // RCC - HSE PLL 72 MHz  
    RCC_DeInit();  
    RCC_HSEConfig(RCC_HSE_ON);  
    ErrorStatus HSEStartUpStatus;  
    HSEStartUpStatus = RCC_WaitForHSEStartUp();  
    if (HSEStartUpStatus == SUCCESS) {  
        FLASH_SetLatency(FLASH_Latency_2);  
        FLASH_PrefetchBufferCmd(FLASH_PrefetchBuffer_Enable);  
        RCC_PCLK1Config(RCC_HCLK_Div2); // PCLK1 Max 36MHz  
        RCC_PLLConfig(RCC_PLLSource_HSE_Div1, RCC_PLLMul_6);  
        RCC_PLLCmd(ENABLE);  
        RCC_SYSCLKConfig(RCC_SYSCLKSource_PLLCLK);  
    } else while(1);  
    while (RCC_GetSYSCLKSource() != 0x08);  
    while (RCC_GetFlagStatus(RCC_FLAG_PLLRDY) == RESET);  
}
```


Configurações – Timer

- Timer 3: período de 500ms (2Hz)
- Modo: contador crescente

```
void Timer_Config() {  
    // Timer 3 - Up Count 500ms  
    TIM_TimeBaseInitTypeDef TIM_TimeBaseStructure;  
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM3, ENABLE);  
    TIM_TimeBaseStructure.TIM_Period = 4000;  
    TIM_TimeBaseStructure.TIM_ClockDivision = TIM_CKD_DIV1;  
    TIM_TimeBaseStructure.TIM_Prescaler = 9000;  
    TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;  
    TIM_TimeBaseInit(TIM3, &TIM_TimeBaseStructure);  
    TIM_Cmd(TIM3, ENABLE);  
}
```

Configurações – USART

- USART2: ligação assíncrona ao computador (via USB)
- *Baud Rate*: 9600 bps

```
void USART_Config() {  
    // USART 2 - 9600bps  
    USART_InitTypeDef USART_InitStructure;  
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_USART2, ENABLE);  
    USART_InitStructure.USART_BaudRate = 9600;  
    USART_InitStructure.USART_WordLength = USART_WordLength_8b;  
    USART_InitStructure.USART_StopBits = USART_StopBits_1;  
    USART_InitStructure.USART_Parity = USART_Parity_No;  
    USART_InitStructure.USART_HardwareFlowControl = USART_HardwareFlowControl_None;  
    USART_InitStructure.USART_Mode = USART_Mode_Tx;  
    USART_Init(USART2, &USART_InitStructure);  
    USART_Cmd(USART2, ENABLE);  
}
```

Configurações – Interrupções

- Configurações:
 - Grupo de Prioridades 1
 - *Update Event* do Timer 3
 - Interrupção Externa 1

```
void NVIC_Config() {  
    // Configuração das Prioridades Group 1  
    NVIC_InitTypeDef NVIC_InitStructure;  
    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_1);  
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;  
    // TIM3  
    NVIC_InitStructure.NVIC_IRQChannel = TIM3_IRQn;  
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;  
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;  
    NVIC_Init(&NVIC_InitStructure);  
    // EXTI1  
    NVIC_InitStructure.NVIC_IRQChannel = EXTI1_IRQn;  
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;  
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 1;  
    NVIC_Init(&NVIC_InitStructure);  
  
    // Ativar a interrupção do Update Event do TIM3  
    TIM_ITConfig(TIM3, TIM_IT_Update, ENABLE);  
  
    // Configuração da interrupção EXTI1  
    GPIO_EXTILineConfig(GPIO_PortSourceGPIOA, GPIO_PinSource1);  
    EXTI_InitTypeDef EXTI_InitStructure;  
    EXTI_InitStructure.EXTI_Line = EXTI_Line1;  
    EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;  
    EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Falling;  
    EXTI_InitStructure.EXTI_LineCmd = ENABLE;  
    EXTI_Init(&EXTI_InitStructure);  
}
```

Configurações – Interrupções

- Rotina da interrupção:
 - A cada 500 ms
 - Contagem de segundos
 - LEDs (ecrã inicial e final)

```
void TIM3_IRQHandler(void) {
    TIM_ClearITPendingBit(TIM3, TIM_IT_Update);
    // Atualizar a informação do acelerómetro no LCD e USART
    flag.updateAccInfo = 1;
    // Contar o tempo do nível (apenas quando o jogo não está em pausa)
    if (flag.gamePaused == 0) {
        tim3_cnt++;
        if (tim3_cnt == 2) {
            t_ls++;
            tim3_cnt = 0;
        }
    }
    // LEDs da Vitória!
    if (flag.gameEnded == 1 && flag.gameFirstStart == 0) {
        if (GPIO_ReadInputDataBit(GPIOB, GPIO_Pin_0) == 0) {
            GPIO_WriteBit(GPIOB, GPIO_Pin_0 | GPIO_Pin_1 | GPIO_Pin_2, Bit_SET);
        } else GPIO_WriteBit(GPIOB, GPIO_Pin_0 | GPIO_Pin_1 | GPIO_Pin_2, Bit_RESET);
    }
    // LEDs aleatórios no Ecrã Inicial
    if (flag.gameFirstStart == 1) {
        if (GPIO_ReadInputDataBit(GPIOB, GPIO_Pin_0) == 0) {
            GPIO_WriteBit(GPIOB, GPIO_Pin_0, Bit_SET);
            GPIO_WriteBit(GPIOB, GPIO_Pin_2, Bit_SET);
        }
        (...)
    }
}
```

Configurações – Interrupções

- Rotina da interrupção:
 - *Set da flag* do botão SW5
 - *Debug* (botão pressionado)

```
void EXTI1_IRQHandler(void) {  
    flag.pressedSW5 = 1;  
    while(GPIO_ReadInputDataBit(GPIOA, GPIO_Pin_1) == 0); // Debug do switch  
    EXTI_ClearITPendingBit(EXTI_Line1);  
}
```

Configurações – SPI

- Comunicação *Full Duplex*
- Placa *Master*
- *Read High*

```
void SPI_Config() {  
    // SPI2 - Full Duplex, Placa Master, Read High  
    SPI_InitTypeDef SPI_InitStructure;  
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_SPI2, ENABLE);  
    SPI_InitStructure.SPI_Direction = SPI_Direction_2Lines_FullDuplex;  
    SPI_InitStructure.SPI_Mode = SPI_Mode_Master;  
    SPI_InitStructure.SPI_DataSize = SPI_DataSize_8b;  
    SPI_InitStructure.SPI_CPOL = SPI_CPOL_High;  
    SPI_InitStructure.SPI_CPHA = SPI_CPHA_2Edge;  
    SPI_InitStructure.SPI_NSS = SPI_NSS_Soft;  
    SPI_InitStructure.SPI_BaudRatePrescaler = SPI_BaudRatePrescaler_2;  
    SPI_InitStructure.SPI_FirstBit = SPI_FirstBit_MSB;  
    SPI_Init(SPI2, &SPI_InitStructure);  
    SPI_Cmd(SPI2, ENABLE);  
  
    GPIO_WriteBit(GPIOD, GPIO_Pin_2, Bit_SET); // ACC SPI Disable  
}
```

Configurações – Acelerómetro

- Configuração do CTRL_REG1: (por SPI)
 - Sensor e Eixos (x, y, z) ativos
 - *Rate*: 40 Hz
 - *Self-test* desligado

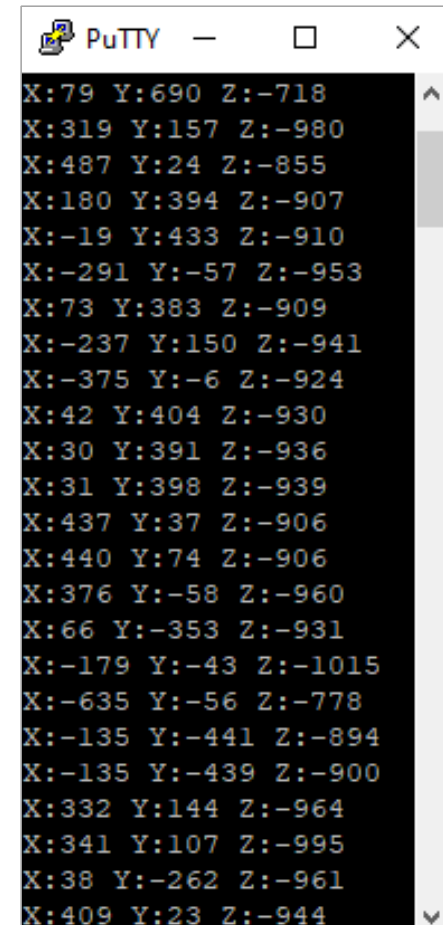
```
void SPI_CREG_Config() {  
    GPIO_WriteBit(GPIOD, GPIO_Pin_2, Bit_RESET); // ACC SPI Enable  
    SPI_Send_Data(0x20); // CTRL_REG1  
    SPI_Send_Data(0b11000111); // Sensor ON, Rate 40Hz, Self Test OFF, Eixos X Y Z ON  
    GPIO_WriteBit(GPIOD, GPIO_Pin_2, Bit_SET); // ACC SPI Disable  
}
```

Leitura SPI

```
uint8_t SPI_Send_Data(uint8_t dataSend) {
    uint8_t dataReceive = 0;
    for(uint8_t i=0;i<2;i++) {
        while (SPI_I2S_GetFlagStatus(SPI2, SPI_I2S_FLAG_TXE) == RESET);
        SPI_I2S_SendData(SPI2, dataSend);
        SPI_I2S_ClearFlag(SPI2, SPI_I2S_FLAG_TXE);

        while (SPI_I2S_GetFlagStatus(SPI2, SPI_I2S_FLAG_RXNE) == RESET);
        dataReceive = SPI_I2S_ReceiveData(SPI2);
        SPI_I2S_ClearFlag(SPI2, SPI_I2S_FLAG_RXNE);
    }
    return dataReceive;
}
```

```
void ACC_Get_Coords() {
    for (uint8_t i=0; i<6; i++) {
        GPIO_WriteBit(GPIOD, GPIO_Pin_2, Bit_RESET); // ACC SPI Enable
        if (i==0) xL = SPI_Send_Data(0b10101000);
        if (i==1) xH = SPI_Send_Data(0b10101001);
        if (i==2) yL = SPI_Send_Data(0b10101010);
        if (i==3) yH = SPI_Send_Data(0b10101011);
        if (i==4) zL = SPI_Send_Data(0b10101100);
        if (i==5) zH = SPI_Send_Data(0b10101101);
        GPIO_WriteBit(GPIOD, GPIO_Pin_2, Bit_SET); // ACC SPI Disable
    }
    ACCx = (xH << 8) | xL;
    ACCy = (yH << 8) | yL;
    ACCz = (zH << 8) | zL;
}
```



PuTTY terminal window showing a list of 30 coordinate pairs (X: Y: Z:). The coordinates are displayed in a monospaced font on a black background. The window title is "PuTTY".

```
X:79 Y:690 Z:-718
X:319 Y:157 Z:-980
X:487 Y:24 Z:-855
X:180 Y:394 Z:-907
X:-19 Y:433 Z:-910
X:-291 Y:-57 Z:-953
X:73 Y:383 Z:-909
X:-237 Y:150 Z:-941
X:-375 Y:-6 Z:-924
X:42 Y:404 Z:-930
X:30 Y:391 Z:-936
X:31 Y:398 Z:-939
X:437 Y:37 Z:-906
X:440 Y:74 Z:-906
X:376 Y:-58 Z:-960
X:66 Y:-353 Z:-931
X:-179 Y:-43 Z:-1015
X:-635 Y:-56 Z:-778
X:-135 Y:-441 Z:-894
X:-135 Y:-439 Z:-900
X:332 Y:144 Z:-964
X:341 Y:107 Z:-995
X:38 Y:-262 Z:-961
X:409 Y:23 Z:-944
```


Algoritmo do Labirinto

- Geração do labirinto:
 - Utiliza o Algoritmo de Prim
 - Tamanho: 21x21
 - Sempre aleatório e possível
 - Início e fim nas mesmas posições
 - *Print* na consola com “# O X”

Open-Source: <https://github.com/fudgenuggets12/Maze-Generator>

```
#####
#O  #           #  #
### ##### # ###
#  #  #  #  #  #
#  ### #  #  #  #
#  #  #  #  #  #
#  ### #  #  #####
#  #  #  #  #  #
#  #  ##### #  ###
#  #  #  #  #  #
##### ### # #####
#  #  #  #  #  #
#  #  #  ##### #  #
#  #  #  #  #  #
#  #  #  #  #  #
#  #  #  #  #  #
#  ### ### #  ###
#  #  #  #  #  #
#  #  ##### ### #
#           #  X#
#####
```

Algoritmo do Labirinto (Modificado)

```
//Prints the finished maze
for(int n = 0; n<21; n++) {
    for(int k = 0; k<21; k++) {
        //No algoritmo original o labirinto seria imprimido na consola
        //printf("%c", maze[n][k]);

        // Impressão no LCD (com scale x6.0 /// +1 => centrar o labirinto no LCD)
        if (maze[n][k] == '#') {
            for(int j = 0; j<6; j++) {
                for(int p = 0; p<6; p++) {
                    lcd_draw_pixel((n*6)+p+1, (k*6)+j+1, RGB_Color_Convert(255, 117, 63));
                    mazeLines[(n*6)+p+1][(k*6)+j+1] = 1;
                }
            }
        }
    }
}

// Abertura no final do labirinto e desenho de uma meta
for (int j=0;j<6;j++) {
    for (int p=0;p<6;p++) {
        mazeLines[114+p+1][120+j+1] = 0;
        if (j == 3 || j == 5) {
            if (p == 1 || p == 3 || p == 5) {
                lcd_draw_pixel(114+p+1, 120+j+1, 0);
            } else lcd_draw_pixel(114+p+1, 120+j+1, RGB_Color_Convert(200, 200, 200));
        } else {
            if (j == 4) {
                if (p == 0 || p == 2 || p == 4) {
                    lcd_draw_pixel(114+p+1, 120+j+1, 0);
                } else lcd_draw_pixel(114+p+1, 120+j+1, RGB_Color_Convert(200, 200, 200));
            } else lcd_draw_pixel(114+p+1, 120+j+1, 0);
        }
    }
}
```



Jogo – *Maze Runner*

- Função *Start_Game()*:
 - Limpar LCD
 - *Reset* dos LEDs
 - *Countdown* (3, 2, 1, GO!)
 - Iniciar o nível

```
void Start_Game() {  
    // Reset LEDs e limpar LCD  
    flag.gamePaused = 0;  
    GPIO_WriteBit(GPIOB, GPIO_Pin_0 | GPIO_Pin_1 | GPIO_Pin_2, Bit_RESET);  
    lcd_draw_fillrect(0, 0, 128, 160, 0);  
  
    // Wait 1 segundo  
    t_ls=0;  
    while (t_ls != 1);  
  
    // Countdown  
    t_ls=0;  
    lcd_draw_string(55, 60, "3", RGB_Color_Convert(255, 171, 63), 3);  
    while (t_ls != 1);  
    lcd_draw_fillrect(0, 0, 128, 90, 0);  
    lcd_draw_string(55, 60, "2", RGB_Color_Convert(255, 171, 63), 3);  
    while (t_ls != 2);  
    lcd_draw_fillrect(0, 0, 128, 90, 0);  
    lcd_draw_string(55, 60, "1", RGB_Color_Convert(255, 171, 63), 3);  
    while (t_ls != 3);  
    lcd_draw_fillrect(0, 0, 128, 90, 0);  
    lcd_draw_string(45, 60, "GO!", RGB_Color_Convert(255, 30, 30), 3);  
    while (t_ls != 4);  
  
    // Linha separadora e começar o nível  
    lcd_draw_fillrect(0, 148, 128, 1, RGB_Color_Convert(255, 30, 30));  
    p.level=1;  
    Start_Level();  
}
```

Jogo – *Maze Runner*

- Função *Start_Level()*:
 - Limpar LCD
 - Desenhar o labirinto
 - Verificações do nível (guardar tempos, acender LEDs...)
 - Retomar a posição inicial

```
void Start_Level() {  
    // Nível 4 => Fim do Jogo (guarda o tempo e não precisa de executar o resto)  
    if (p.level == 4) {  
        p.level_time[2] = t_ls;  
        End_Game();  
        return;  
    }  
  
    // Limpar LCD e desenhar novo labirinto  
    lcd_draw_fillrect(0, 0, 128, 146, 0);  
    Draw_Maze();  
  
    // Nível 1, 2 e 3 - Guardar Tempo e mostrar info nos LEDs  
    if (p.level == 1) GPIO_WriteBit(GPIOB, GPIO_Pin_0, Bit_SET);  
    if (p.level == 2) {  
        p.level_time[0] = t_ls;  
        GPIO_WriteBit(GPIOB, GPIO_Pin_1, Bit_SET);  
    }  
    if (p.level == 3) {  
        p.level_time[1] = t_ls;  
        GPIO_WriteBit(GPIOB, GPIO_Pin_2, Bit_SET);  
    }  
    // Mostrar info no LCD  
    char buffer[32];  
    sprintf(buffer, "LEVEL %d", p.level);  
    lcd_draw_string(1, 138, buffer, RGB_Color_Convert(255, 30, 30), 1);  
  
    // Retomar a posição inicial e reset no tempo  
    p.x = 10; p.y = 10;  
    lcd_draw_filled_circle(p.x, p.y, 2, RGB_Color_Convert(255, 171, 63));  
    t_ls = 0;  
}
```

Jogo – *Maze Runner*

- Função *Move_Player()*:
 - Armazenar a posição anterior
 - Verificar as paredes do labirinto
 - Se permitido, desenhar a nova posição
 - Verificar se chegou ao fim do labirinto

```
void Move_Player() {  
    // Guardar a posição anterior  
    p.previous_x = p.x;  
    p.previous_y = p.y;  
  
    // Nova posição  
    if(ACCx >= 170) p.x+=1;  
    if(ACCx <= -170) p.x-=1;  
    if(ACCy >= 170) p.y+=1;  
    if(ACCy <= -170) p.y-=1;  
  
    // Verificar as paredes do labirinto (caso seja uma parede => manter a posição anterior)  
    if (mazeLines[p.x+1][p.y+2] == 1 || mazeLines[p.x+2][p.y+1] == 1 ||  
        mazeLines[p.x+1][p.y-2] == 1 || mazeLines[p.x+2][p.y-1] == 1 ||  
        mazeLines[p.x-1][p.y+2] == 1 || mazeLines[p.x-2][p.y+1] == 1 ||  
        mazeLines[p.x-1][p.y-2] == 1 || mazeLines[p.x-2][p.y-1] == 1) {  
  
        p.x = p.previous_x;  
        p.y = p.previous_y;  
    }  
  
    // Não executar o resto se a posição nova for igual à anterior  
    if (p.x == p.previous_x && p.y == p.previous_y) return;  
  
    // Desenhar o player na nova posição (bola amarela)  
    lcd_draw_filled_circle(p.previous_x, p.previous_y, 2, 0);  
    lcd_draw_filled_circle(p.x, p.y, 2, RGB_Color_Convert(255, 171, 63));  
  
    // Verificar se é o final do labirinto e começar o nível seguinte (ou terminar o jogo)  
    if ((p.x == 117 || p.x == 118) && p.y == 123) {  
        p.level++;  
        Start_Level();  
    }  
}
```

Jogo – *Maze Runner*

- Função *End_Game()*:
 - Limpar LCD
 - Apresentação do Menu Final
 - Exibir os tempos em cada nível e total acumulado

```
void End_Game() {
    flag.gameEnded = 1;

    // Limpar LCD
    lcd_draw_fillrect(0, 0, 128, 170, 0);

    // Menu Final
    lcd_draw_string(7, 11, "GOOD GAME!", RGB_Color_Convert(255, 171, 63), 2);
    lcd_draw_string(34, 40, "SCORE", RGB_Color_Convert(255, 117, 63), 2);
    lcd_draw_string(10, 135, "AGAIN:SW5", RGB_Color_Convert(255, 30, 30), 2);

    // Mostrar os tempos de todos os niveis e o total acumulado
    uint8_t time_min;
    uint8_t time_seg;
    char time_str[32];
    for (uint8_t i=0;i<3;i++) {
        time_min = p.level_time[i] / 60;
        time_seg = p.level_time[i] % 60;
        if (time_min < 10) {
            if (time_seg >= 10) sprintf(time_str, "LEVEL %d 0d:%d", i+1, time_min, time_seg);
            if (time_seg < 10) sprintf(time_str, "LEVEL %d 0d:0%d", i+1, time_min, time_seg);
        } else {
            if (time_seg >= 10) sprintf(time_str, "LEVEL %d %d:%d", i+1, time_min, time_seg);
            if (time_seg < 10) sprintf(time_str, "LEVEL %d %d:0%d", i+1, time_min, time_seg);
        }
        lcd_draw_string(21, 65+i*15, time_str, RGB_Color_Convert(255, 117, 63), 1);
    }
    time_min = (p.level_time[0] + p.level_time[1] + p.level_time[2]) / 60;
    time_seg = (p.level_time[0] + p.level_time[1] + p.level_time[2]) % 60;
    if (time_min < 10) {
        if (time_seg >= 10) sprintf(time_str, "TOTAL 0d:%d", time_min, time_seg);
        if (time_seg < 10) sprintf(time_str, "TOTAL 0d:0%d", time_min, time_seg);
    } else {
        if (time_seg >= 10) sprintf(time_str, "TOTAL %d:%d", time_min, time_seg);
        if (time_seg < 10) sprintf(time_str, "TOTAL %d:0%d", time_min, time_seg);
    }
    lcd_draw_string(33, 110, time_str, RGB_Color_Convert(255, 117, 63), 1);
}
```

Jogo – *Maze Runner*

- Função *Update_Level_Time()*:
 - Atualizar o tempo decorrido durante o nível

```
void Update_Level_Time() {  
    // Atualizar o tempo do nível decorrente (LCD)  
    uint8_t time_min = t_ls / 60;  
    uint8_t time_seg = t_ls % 60;  
    char time_str[16];  
    if (time_min < 10) {  
        if (time_seg >= 10) sprintf(time_str, "TIME 0%d:%d", time_min, time_seg);  
        if (time_seg < 10) sprintf(time_str, "TIME 0%d:0%d", time_min, time_seg);  
    } else {  
        if (time_seg >= 10) sprintf(time_str, "TIME %d:%d", time_min, time_seg);  
        if (time_seg < 10) sprintf(time_str, "TIME %d:0%d", time_min, time_seg);  
    }  
    lcd_draw_string(68, 138, time_str, RGB_Color_Convert(255, 30, 30), 1);  
}
```

Funções Auxiliares

- Função *Update_ACC_Info()*:
 - Valores do acelerómetro (x, y, z)
 - Execução: a cada 500 ms
 - Envio para USART
 - *Display* no LCD

```
void Update_ACC_Info() {  
    // Mostrar os valores do acelerómetro (x, y, z) no LCD e na USART  
    char buffer[32];  
  
    // USART  
    sprintf(buffer, "X:%d Y:%d Z:%d\r\n", ACCx, ACCy, ACCz);  
    USART2_SendMessage(buffer);  
  
    // LCD  
    char eraseNumber[5] = "";  
    char coordACC[3] = {'X', 'Y', 'Z'};  
    int16_t ACC[3] = {ACCx, ACCy, ACCz};  
    for (uint8_t i=0; i<3; i++) {  
        if (ACC[i] >= 0) strcpy(eraseNumber, " ");  
        if (abs(ACC[i]) < 1000) strcpy(eraseNumber, " ");  
        if (abs(ACC[i]) < 100) strcpy(eraseNumber, " ");  
        if (abs(ACC[i]) < 10) strcpy(eraseNumber, " ");  
        sprintf(buffer, "%c:%d%s", coordACC[i], ACC[i], eraseNumber);  
        lcd_draw_string(43*i, 152, buffer, RGB_Color_Convert(255, 30, 30), 1);  
    }  
}
```


Funções Auxiliares

- Função *USART2_SendMessage()*:
 - Envio de uma mensagem para o USART2

```
void USART2_SendMessage(char *message) {  
    uint16_t aux=0;  
    while(aux != strlen(message)) {  
        USART_SendData(USART2, (uint8_t) message[aux]);  
        while(USART_GetFlagStatus(USART2, USART_FLAG_TXE) == RESET);  
        aux++;  
    }  
}
```

Funções Auxiliares

- Função *RGB_Color_Convert()*:
 - Conversão de cor RGB de 24 bits para 16 bits

```
uint16_t RGB_Color_Convert(char r, char g, char b) {  
    // Conversão RGB 24 bits para RGB 16 bits (R: 5 bits / G: 6 bits / B: 5 bits)  
    r = (r*31)/255;  
    g = (g*63)/255;  
    b = (b*31)/255;  
    return ((r) | (g << 5) | (b << 11));  
}
```

Função Principal – *main()*

- Inicializações e configurações
- Apresentação do Menu Inicial

```
int main(void) {  
    // Inicializações e Configurações  
    RCC_Config_HSE_PLL_Max();  
    lcd_init();  
    GPIO_Config();  
    SPI_Config();  
    USART_Config();  
    Timer_Config();  
    NVIC_Config();  
    SPI_SREG_Config();  
  
    // Configurações Iniciais das Flags  
    flag.gameFirstStart = 1;  
    flag.gamePaused = 1;  
    flag.gameEnded = 1;  
  
    // Menu Inicial  
    lcd_draw_string(26, 30, "MAZE", RGB_Color_Convert(255, 171, 63), 3);  
    lcd_draw_string(10, 60, "RUNNER", RGB_Color_Convert(255, 171, 63), 3);  
    lcd_draw_string(10, 110, "START:SW5", RGB_Color_Convert(255, 30, 30), 2);  
}
```

Função Principal – *main()*

- Execução repetida *while(1)*:
 - Verificação do botão SW5
 - Iniciar, pausar ou retomar o jogo

```
// SW5: Inicar, pausar ou retomar o Jogo
if (flag.pressedSW5 == 1) {
    flag.pressedSW5 = 0; // Debug do switch

    if (flag.gameEnded == 0) {
        // Pausar o Jogo
        if (flag.gamePaused == 0) {
            flag.gamePaused = 1;
            // Escreve "PAUSE" em cima do labirinto
            lcd_draw_fillrect(17, 52, 94, 28, 0);
            lcd_draw_string(20, 56, "PAUSA", RGB_Color_Convert(255, 30, 30), 3);

            // Retomar o Jogo
        } else {
            flag.gamePaused = 0;
            // Redesenhar o player e o labirinto na parte onde foi escrito "PAUSE"
            for (uint8_t x=16; x<=112; x++) {
                for (uint8_t y=51; y<=81; y++) {
                    if (mazeLines[x][y] == 1) {
                        lcd_draw_pixel(x, y, RGB_Color_Convert(255, 117, 63));
                    } else lcd_draw_pixel(x, y, 0);
                }
            }
            lcd_draw_filled_circle(p.x, p.y, 2, RGB_Color_Convert(255, 171, 63));
        }
    }

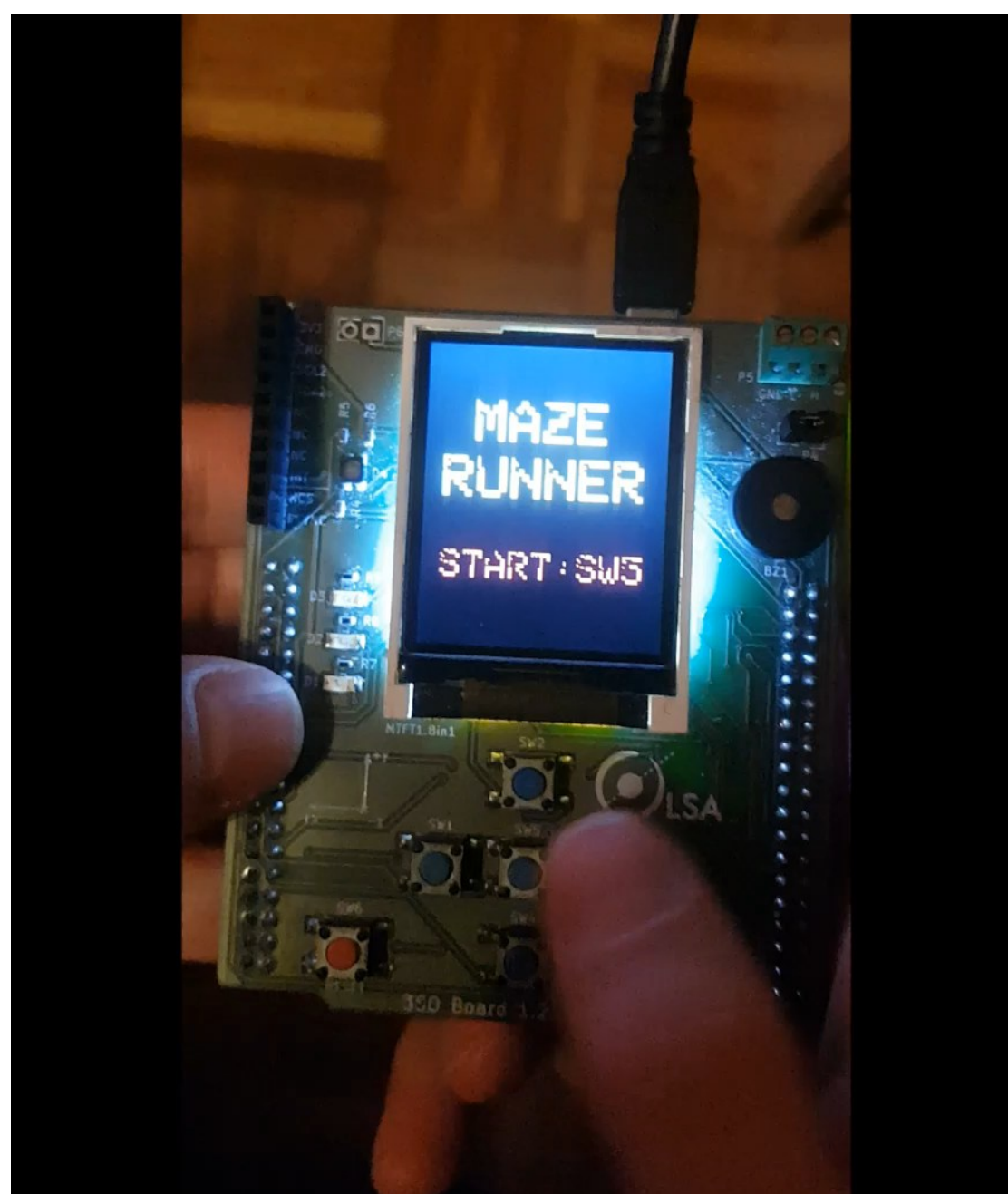
    // Iniciar o Jogo (ou re-iniciar após vitória)
    if (flag.gameEnded == 1) {
        flag.gameFirstStart = 0;
        flag.gameEnded = 0;
        Start_Game();
    }
}
```

Função Principal – *main()*

- Execução repetida *while(1)*:
 - Obter coordenadas do acelerómetro
 - Jogar!

```
// Se não estiver no Menu Inicial ou no Fim do Jogo
if (flag.gameEnded == 0) {
    // Obter as coordenadas do acelerómetro
    ACC_Get_Coords();
    // Atualizar informação do acelerómetro no LCD e USART (a cada 500ms)
    if (flag.updateAccInfo == 1) {
        flag.updateAccInfo = 0;
        Update_ACC_Info();
    }
    // Em jogo => Atualizar o tempo de nível decorrente e mover o player no labirinto
    if (flag.gamePaused == 0) {
        Update_Level_Time();
        Move_Player();
    }
}
```

Demonstração



Obrigado pela atenção!

