ISEL
Departamento de Engenharia Eletrónica e Telecomunicações e de Computadores

Licenciatura em Engenharia Informática e de Computadores

Jogo Invasores Espaciais (*Space Invaders Game*)

Jorge Albino
Duarte Patrício
Lourenço Teles

Projeto
de
Laboratório de Informática e Computadores
2023 / 2024
Verão

13 de junho de 2024

1 INTRODUÇÃO 1

2 ARQUITETURA DO SISTEMA 2

3 IMPLEMENTAÇÃO DO SISTEMA 3

4 CONCLUSÕES 3

A. INTERLIGAÇÕES ENTRE O HW E SW 5

B. ATRIBUIÇÃO DE PINOS 6

C. CÓDIGO KOTLIN *HAL* 8

D. CÓDIGO KOTLIN *KBD* 9

E. CÓDIGO KOTLIN *LCD* 10

F. CÓDIGO KOTLIN *SERIALEMITTER* 13

G. CÓDIGO KOTLIN *SCOREDISPLAY* 14

H. CÓDIGO KOTLIN *TUI* 16

***I.* CÓDIGO KOTLIN *M* 18**

***J.* CÓDIGO KOTLIN *COINACCEPTOR* 19**

***K.* CÓDIGO KOTLIN *FILEACCESS* 20**

***L.* CÓDIGO KOTLIN *SCORES* 21**

***M.* CÓDIGO KOTLIN *STATISTICS* 23**

N. CÓDIGO KOTLIN *APP* 24

1 Introdução

Neste projeto implementa-se o jogo Invasores Espaciais (*Space Invaders Game*) utilizando um PC e periféricos para interação com o jogador. Neste jogo, os invasores espaciais são representados por números entre 0 e 9, e a nave espacial realiza mira sobre o primeiro invasor da fila eliminando-o, se no momento do disparo os números da mira e do invasor coincidirem. O jogo termina quando os invasores espaciais atingirem a nave espacial. Para se iniciar um jogo é necessário um crédito, obtido pela introdução de moedas. O sistema só aceita moedas de 1,00€, que correspondem a dois créditos.
O sistema de jogo é constituído por: um teclado de 12 teclas; um moedeiro (*Coin Acceptor*); um mostrador *Liquid Cristal Display* (*LCD*) de duas linhas com 16 caracteres cada; um mostrador de pontuação (*Score Display*) e uma chave de manutenção designada por *M*, para colocação do sistema em modo de Manutenção. O diagrama de blocos do jogo Invasores Espaciais é apresentado na Figura 1.

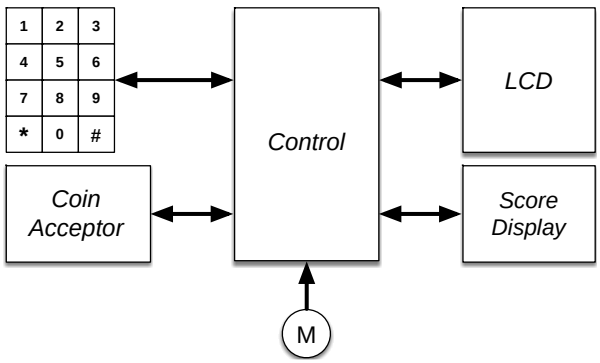

Figura 1 – Diagrama de blocos do jogo Invasores Espaciais (*Space Invaders Game*)

Sobre o sistema proposto podem realizar-se as seguintes ações em modo de Jogo:

- **Jogo –** O jogo inicia-se quando for premida a tecla '**#**' e existirem créditos disponíveis. Os Invasores Espaciais aparecem do lado direito do *LCD*, em ambas as linhas. Ao premir a tecla '*' a mira do canhão da nave permuta de linha, utilizando as teclas numéricas (0-9) efetua-se a mira sobre o invasor sendo este eliminado após a realização do disparo que é executado quando for premida a tecla '**#**'. O jogo termina quando os invasores atingirem a nave espacial. A pontuação final é determinada pelo acumular dos pontos realizados durante o jogo, estes são obtidos através da eliminação dos invasores.
- **Visualização da Lista de Pontuações –** Esta ação é realizada sempre que o sistema está modo de espera de início de um novo jogo e após a apresentação, por 6 segundos da mensagem de identificação do jogo.

No modo Manutenção podem realizar-se as seguintes ações sobre o sistema:

- **Teste** – Permite realizar um jogo, sem créditos e sem a pontuação do jogo ser contabilizada para a Lista de Pontuações.
- **Consultar os contadores de moedas e jogos** – Carregando na tecla '**#**' permite-se a listagem dos contadores de moedas e jogos realizados.
- **Reiniciar os contadores de moedas e jogos –** Premindo a tecla '**#**' e em seguida a tecla '*****', o sistema de gestão coloca os contadores de moedas e jogos a zero, iniciando um novo ciclo de contagem.
- **Desligar –** Permite desligar o sistema, que encerra apenas após a confirmação do utilizador, ou seja, o programa termina e as estruturas de dados, contendo a informação dos contadores e da Lista de Pontuações, são armazenadas de forma persistente em dois ficheiros de texto, por linha e com os campos de dados separados por ";". O primeiro ficheiro deverá conter o número de jogos realizados e o número de moedas guardadas no cofre do moedeiro. O segundo ficheiro deverá conter a Lista de Pontuações, que compreende as 20 melhores pontuações e o respetivo nome do jogador. Os dois ficheiros devem ser carregados para o sistema no seu processo de arranque.

2 Arquitetura do sistema

O sistema é implementado numa solução híbrida de *hardware* e *software*, como apresentado no diagrama de blocos da Figura 2. A arquitetura proposta é constituída por cinco módulos principais: *i*) um leitor de teclado, designado por *Keyboard Reader*; *ii*) um módulo de interface com o *LCD*, designado por *Serial LCD Controller* (*SLCDC*); *iii*) um módulo de interface com o mostrador de pontuação (*Score Display*), designado por *Serial Score Controller* (*SSC*); *iv*) um moedeiro, designado por *Coin Acceptor*; e *v*) um módulo de controlo, designado por *Control*. Os módulos *i*), *ii*) e *iii*) são implementados em *hardware*, o moedeiro é simulado, enquanto o módulo de controlo é implementado em *software*, executado num PC usando linguagem *Kotlin*.

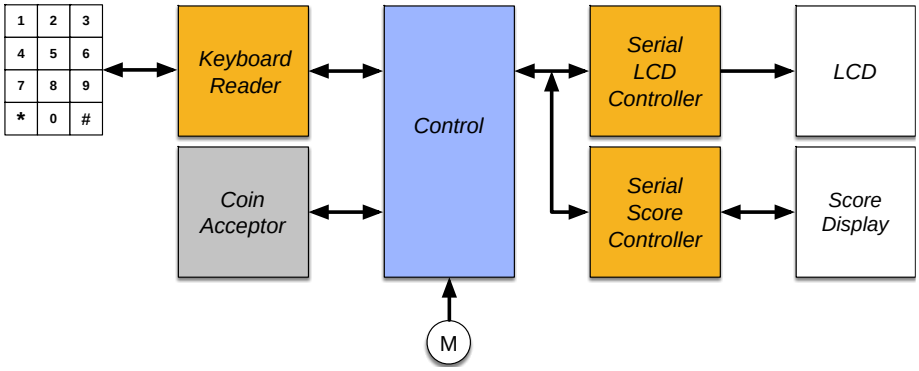

Figura 2 – Arquitetura do sistema que implementa o jogo Invasores Espaciais (*Space Invaders Game*)

O módulo *Keyboard Reader* é responsável pela descodificação do teclado matricial de 12 teclas, determinando qual a tecla pressionada e disponibilizando o seu código, com quatro bits, ao módulo *Control*. Caso este não esteja disponível para o receber imediatamente, o código da tecla é armazenado até ao limite de dez códigos. O módulo *Control* processa os dados e envia a informação a apresentar no *LCD* através do módulo *SLCDC*. O mostrador de pontuação é atuado pelo módulo *Control*, através do módulo *SSC*. Por razões de ordem física, e por forma a minimizar o número de interligações, a comunicação entre o módulo *Control* e os módulos *SLCDC e SSC* é realizada através de um protocolo série síncrono.
A implementação do módulo *Control* foi realizada em *software*, usando a linguagem *Kotlin* e seguindo a arquitetura lógica apresentada na Figura 3.

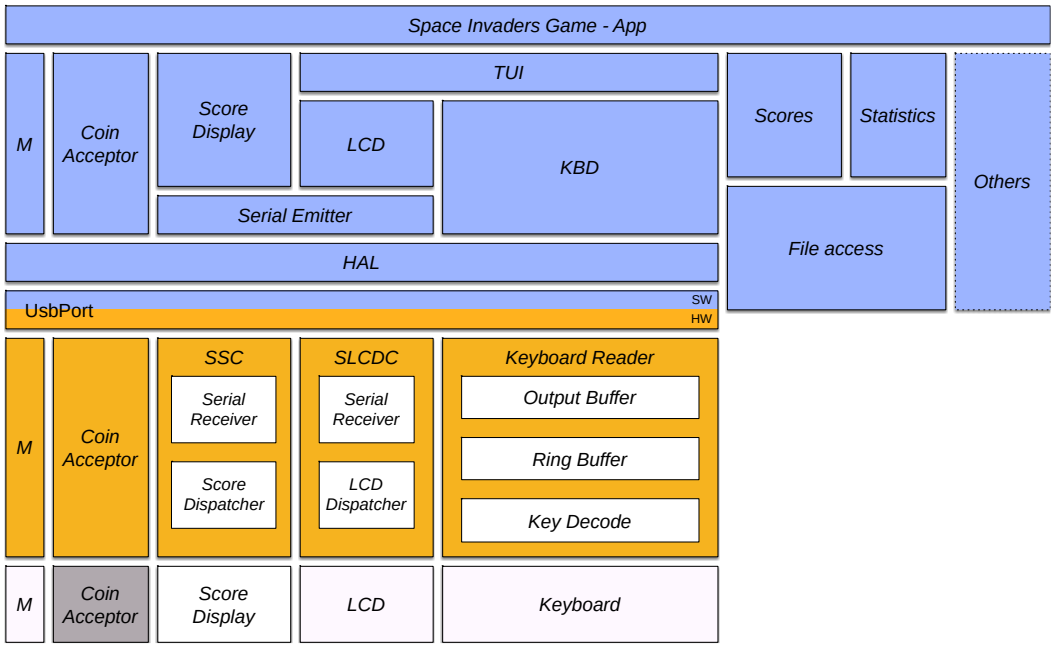

Figura 3 – Diagrama lógico do Jogo Invasores Espaciais (*Space Invaders Game*)

3 Implementação do sistema

O sistema, tal como dito supra, é implementado numa solução híbrida de *Hardware* e *Software*, tendo uma estrutura envolvente em cinco módulos principais de *Hardware* e doze módulos principais de *Software*.

Quanto ao módulo *KeyBoard Reader* este compreende mais 3 módulos dentro dele, o *Output Buffer*, o *Ring Buffer* e o *Key Decode*. O *Output Buffer* guarda um dos dez códigos aos quais o modulo *KeyBoard Reader* suporta armazenar, este módulo é responsável pelo envio direto para o *Software*, estando diretamente conectado com o módulo *Control*. O *Ring Buffer* é responsável pelo armazenamento de outros 8 códigos, guardando-os numa memória de oito endereços, quando o módulo *Output Buffer* envia um sinal de prontidão a receber mais um código, este módulo envia o primeiro ao qual foi guardado na RAM, tendo uma filosofia FIFO (*First In First Out*). O *Key Decode* decodifica os sinais recebidos do modulo de *Hardware KeyBoard*, utilizando a técnica de varrimento como verificação de teclas pressionadas.

Quanto ao módulo *SSC* e *SLCDC*, estes responsabilizam-se pela receção de informação do *Control* para os componentes *LCD* e *Score Display*, tendo integrados dois módulos, o *Serial Receiver* e o *Score/LCD Dispatcher*. O *Serial Receiver* recebe os dados enviados através de um protocolo série síncrono e junta-os para que possa enviar para os componentes uma sequência de dados em paralelo. Os *Scores/LCD Dispatcher* são responsáveis pela entrega de tramas válidas para os seus respetivos componentes. A diferença entre estes dois módulos gerais é a quantidade de bits que estão capazes de receber, o *LCD* pretende receber 9 bits de dados e o bit de paridade, o *Score Display* pretende receber 9 bits de dados e o bit de paridade.

Quanto aos outros dois módulos observados na figura 3, o *Coin Acceptor* e o *M* são responsáveis pelo envio de sinal da inserção de uma moeda e da entrada no modo de manutenção da aplicação, respetivamente. O *Coin Acceptor* apenas coloca o seu sinal a '0' lógico, uma vez que receba um sinal *accept* do módulo *Control*.

Os módulos de *software* são maioritariamente dependentes do *HAL* (*Hardware Abstraction Layer*), sendo este o módulo de receção e envio do *Software*. O módulo *KBD* é atribuído à tradução dos sinais recebidos pelo *HAL*, oriundos do teclado. O módulo *LCD* e o módulo *Score Display* têm como função enviar instruções ao *hardware* via o módulo *Serial Emitter* para este o organizar e enviar para o *LCD* e para o *Score Display*. O *Serial Emitter* corresponde ao objeto que executa o protocolo de envio em série síncrono. O módulo *TUI* (*Text-Based User Interface*) junta os módulos *LCD* e *KBD* criando uma interação par ação-reação referente ao teclado e ao *LCD*. Os módulos *Coin Acceptor* e *M* do *software* têm a função de ler o bit de máscara dos respetivos sinais enviados pelo *hardware*. Os módulos não dependentes do *HAL*, *File Access*, *Scores* e *Statistics* são responsáveis pela leitura, escrita e organização dos contadores de *coins* e *games*. O módulo *APP* conjuga os módulos *TUI*, *M*, *Coin Acceptor*, *Score Display*, *Scores* e *Statistics*, tendo como função implementar o jogo *Space Invaders*.

4 Conclusões

Neste projeto, implementou-se o jogo Invasores Espaciais utilizando um PC e periféricos para interação com o jogador. O sistema proposto oferece uma experiência interativa e desafiadora, onde os invasores são representados por números e o jogador deve alinhar a mira para eliminá-los. O jogo requer a inserção de moedas para iniciar, oferecendo dois créditos por cada moeda de 1,00€. Diversos componentes, como um teclado de 12 teclas detetáveis, um moedeiro, um LCD de duas linhas, um mostrador de pontuação e uma chave de manutenção, foram integrados para formar o sistema de jogo.

Durante o modo de jogo, o jogador pode iniciar o jogo, trocar a linha de mira, e disparar para eliminar invasores, com a pontuação sendo acumulada conforme o desempenho. No modo de manutenção, funcionalidades adicionais permitem testar o jogo, consultar e reiniciar contadores de moedas e jogos, e desligar o sistema de forma segura, garantindo a persistência dos dados de contadores e pontuações.

Com a implementação completa, o sistema oferece uma experiência de jogo completa e funcional, com um mecanismo de manutenção robusto e armazenamento persistente de dados, proporcionando entretenimento e desafio contínuos para os jogadores.

A. Interligações entre o HW e SW

Para fazer as interligações, teremos de ir ao ficheiro "Space_Invaders.vhd", pois será esse a entidade de topo onde teremos de fazer as ligações iniciais, ou seja, as entradas que irão começar a transferência de tramas e bits no hardware e onde terá de se conectar com o software. Para chegarmos a esse fim, de fazer a conexão entre o software e hardware, teremos de usar a componente "USBport". As seguintes conexões terão uma explicação depois do código:

***Hardware* para *Software*:**

- Os primeiros 4 bits menos significativos utilizados para a troca de códigos do *KeyboardReader* para o *KBD* (passando pelo módulo *HAL*);
- O quinto bit menos significativos serve para o envio do sinal de aviso da existência de um código de tecla à espera de ser lido;
- O sétimo bit envia o sinal do *Coin Acceptor* para o *software*;
- O oitavo bit envia o sinal de entrada de manutenção para o *software*.

Software para Hardware

- Os primeiros 2 bits servem para os sinais de *SS* (*Slave Selector*), sinais de seleção de ativação entre o *SSC* e o *SLCDC*;
- O quarto bit é onde é enviado o bit de dados que o *Serial Emitter* produz;
- O quinto bit transmite o sinal de relógio do *Serial Emitter*;
- O sétimo bit envia o sinal de *accept* ao módulo *Coin Acceptor* do *Hardware*;
- O oitavo bit envia o sinal de *acknowledge* ao módulo *Keyboard Reader*, quando este processa o código de tecla enviado pelo *Hardware*.

Como se pode observar no código, todas as entradas e saídas do USBport estarão conectadas a um sinal de que por sua vez estarão ligados à entrada/saída que correspondem, por exemplo: Keyboard_Data_to_Software (um vector de 4 bits) estará ligado ao Final_Data (a saída do keyboard_reader).

B. Atribuição de Pinos

PIN_V5 -to LCD_EN

PIN_W8 -to LCD_OUT [0]

PIN_AA15 -to LCD_OUT [1]

PIN_W13 -to LCD_OUT[2]

PIN_AB13 -to LCD_OUT[3]

PIN_Y11 -to LCD_OUT[4]

PIN_W11 -to LCD_OUT[5]

PIN_AA10 -to LCD_OUT[6]

PIN_Y8 -to LCD_OUT[7]

PIN_Y7 -to LCD_OUT[8]

PIN_A8 -to accept_Signal

PIN_C14 -to HEX0[0]

PIN_E15 -to HEX0[1]

PIN_C15 -to HEX0[2]

PIN_C16 -to HEX0[3]

PIN_E16 -to HEX0[4]

PIN_D17 -to HEX0[5]

PIN_C17 -to HEX0[6]

PIN_D15 -to HEX0[7]

PIN_C18 -to HEX1[0]

PIN_D18 -to HEX1[1]

PIN_E18 -to HEX1[2]

PIN_B16 -to HEX1[3]

PIN_A17 -to HEX1[4]

PIN_A18 -to HEX1[5]

PIN_B17 -to HEX1[6]

PIN_A16 -to HEX1[7]

PIN_B20 -to HEX2[0]

PIN_A20 -to HEX2[1]

PIN_B19 -to HEX2[2]

PIN_A21 -to HEX2[3]

PIN_B21 -to HEX2[4]
PIN_C22 -to HEX2[5]
PIN_B22 -to HEX2[6]
PIN_A19 -to HEX2[7]

PIN_F21 -to HEX3[0]
PIN_E22 -to HEX3[1]
PIN_E21 -to HEX3[2]
PIN_C19 -to HEX3[3]
PIN_C20 -to HEX3[4]
PIN_D19 -to HEX3[5]
PIN_E17 -to HEX3[6]
PIN_D22 -to HEX3[7]

PIN_F18 -to HEX4[0]
PIN_E20 -to HEX4[1]
PIN_E19 -to HEX4[2]
PIN_J18 -to HEX4[3]
PIN_H19 -to HEX4[4]
PIN_F19 -to HEX4[5]
PIN_F20 -to HEX4[6]
PIN_F17 -to HEX4[7]

PIN_J20 -to HEX5[0]
PIN_K20 -to HEX5[1]
PIN_L18 -to HEX5[2]
PIN_N18 -to HEX5[3]
PIN_M20 -to HEX5[4]
PIN_N19 -to HEX5[5]
PIN_N20 -to HEX5[6]
PIN_L19 -to HEX5[7]

PIN_F15 -to General_Reset
PIN_C10 -to Manut
PIN_C11 -to Coin_Inserter

C. Código Kotlin *HAL*

```
import isel.leic.UsbPort

fun main(){
    HAL.init()
}

object HAL { // Virtualiza o acesso ao sistema UsbPort

    var Output = 0


    // Inicia a classe
    fun init() {
        clrBits(0xFF)
    }


    // Retorna true se o bit tiver o valor lógico ‘1’
    fun isBit(mask: Int): Boolean {
        return (readBits(mask) == mask)
    }
    // Retorna os valores dos bits representados por mask presentes no UsbPort
    fun readBits(mask: Int): Int {
        return UsbPort.read().and(mask)
    }
    // Escreve nos bits representados por mask os valores dos bits correspondentes em value
    fun writeBits(mask: Int, value: Int) {
        Output = (value.and(mask)).or(Output.and(mask.inv()))
            UsbPort.write(Output)
    }

    // Coloca os bits representados por mask no valor lógico ‘1’
    fun setBits(mask: Int) {
        Output = Output.or(mask)
        UsbPort.write(Output)
    }

    // Coloca os bits representados por mask no valor lógico ‘0’
    fun clrBits(mask: Int) {
        Output = Output.and(mask.inv())
        UsbPort.write(Output)
    }
}
/*Código feito ao som de:
    goosebumps by Travis Scott
 */
```

D. Código Kotlin *KBD*

```
object KBD {
    const val KVAL_MASK = 0x10
    const val KACK_MASK = 10000000
    const val DATA_MASK = 0x0F
    const val NONE = ' '
// Inicia a classe

    fun init() {
        HAL.clrBits(KACK_MASK) //Garantir que o Kack é 0 para não termos key's a serem acknowledged na
inicialização
    }

    // Retorna de imediato a tecla premida ou NONE se não há tecla premida.
    fun getKey(): Char {

        if (HAL.isBit(KVAL_MASK)) {
            val Read_Input = HAL.readBits(DATA_MASK) //data
            var Key = when (Read_Input) {
                0b0000 -> '1'
                0b0100 -> '2'
                0b1000 -> '3'
                0b0001 -> '4'
                0b0101 -> '5'
                0b1001 -> '6'
                0b0010 -> '7'
                0b0110 -> '8'
                0b1010 -> '9'
                0b0111 -> '0'
                0b1011 -> '#'
                0b0011 -> '*'
                else -> NONE
            }

            HAL.setBits(KACK_MASK) //Se houve input, temos de dar ack
            while(HAL.isBit(KVAL_MASK)); //Esperamos que o Kval vá abaixo
            HAL.clrBits(KACK_MASK)


            return Key
        }
        return NONE
    }

    // Retorna a tecla premida, caso ocorra antes do ‘timeout’ (representado em milissegundos), ou NONE
caso contrário.
    fun waitKey(timeout: Long): Char {
        val init = System.currentTimeMillis()


        while (System.currentTimeMillis() < init + timeout) {
            val Key = getKey()
            if(Key != NONE)
                return Key
        }
        return NONE
    }
}

/*Código feito ao som de:
    Tik Tok by Kesha
 */
```

E. Código Kotlin *LCD*

```
object LCD {
    private const val CLK_REG_MASK = 0x10 // É o bit (0001 0000)
    private const val E_MASK = 0x20 // É o bit 5(0010 0000)
    private const val RS_MASK = 0x40 // É o bit 6 (0100 0000)
    private const val ON_MASK = 0x0C // São os bits 2 e 3(1100)
    private const val OFF_MASK = 0x08 // É o bit 3 (1000)

    private val nave = byteArrayOf( //pixel art
        0b11110,
        0b11000,
        0b11100,
        0b11111,
        0b11100,
        0b11000,
        0b11110,
        0b00000
    )
    private val invader = byteArrayOf( //pixel art
        0b11111,
        0b11111,
        0b10101,
        0b11111,
        0b11111,
        0b10001,
        0b10001,
        0b00000
    )

    // Escreve um byte de comando/dados no LCD em paralelo
    private fun writeByteParallel(rs: Boolean, data: Int) {
        if (rs) {
            HAL.setBits(RS_MASK)
        } else {
            HAL.clrBits(RS_MASK)
        }
        val shift_right = data.shr(4)

        HAL.writeBits(15, shift_right)
        HAL.setBits(CLK_REG_MASK)
        HAL.clrBits(CLK_REG_MASK)

        HAL.writeBits(15, data)
        HAL.setBits(CLK_REG_MASK)
        HAL.clrBits(CLK_REG_MASK)
        HAL.setBits(E_MASK)
        HAL.clrBits(E_MASK)
    }

    // Escreve um byte de comando/dados no LCD em série
    private fun writeByteSerial(rs: Boolean, data: Int) {
        var not_data = data
        if (rs) {
            not_data = data.shl(1) + 1
        } else {
            not_data = data.shl(1)
        }

        SerialEmitter.send(SerialEmitter.Destination.LCD, not_data, 10)
    }

    // Escreve um byte de comando/dados no LCD
    private fun writeByte(rs: Boolean, data: Int) {
```

```
        writeByteSerial(rs, data)
    }

    // Escreve um comando no LCD
    private fun writeCMD(data: Int) {
        writeByte(false, data)
    }

    // Escreve um dado no LCD
    private fun writeDATA(data: Int) {
        writeByte(true, data)
    }

    // Escreve um caráter na posição corrente.
    fun writeChar(c: Char) {
        writeDATA(c.code)
    }

    // Escreve uma string na posição corrente.
    fun writeString(text: String) {
        for (i in text) {
            writeChar(i)
        }
    }

    // Envia comando para posicionar cursor (‘line’:0..LINES-1 , ‘column’:0..COLS-1)
    fun cursor(line: Int, column: Int) {
        val writer = 128
        writeCMD((line * 0x40) + column + writer)
    }

    // Envia comando para limpar o ecrã e posicionar o cursor em (0,0)
    fun clear() {
        writeCMD(0x01)
    }

    // Função usada para definir um caracter especial e guarda-lo na CGRAM
    fun createCustomChar(location: Int, charmap: ByteArray) {
        writeCMD(0x40 or (location shl 3)) // Definir o endereço da CGRAM (3 shifts para multiplica-lo
por 8)

        for (i in charmap.indices) {
            writeDATA(charmap[i].toInt()) //envio de dados sobre o caracter especial
        }
    }

    // Desliga o LCD quando off = true
    fun off(off:Boolean){
        if (off){
            writeCMD(OFF_MASK)
        }
        else
            writeCMD(ON_MASK)
    }

    // Desenha a nave no LCD
    fun drawShip(line: Int, column: Int) {

        createCustomChar(0, nave)
        cursor(line, column) //Aplica o cursor na linha e coluna de input
        writeDATA(0) // Display do caracter guardado na CGram na posição 0
    }
    fun drawInvader(line:Int, column: Int){

        createCustomChar(1, invader)
```

```
        cursor(line, column) //Aplica o cursor na linha e coluna de input
        writeDATA(1) // Display do caracter guardado na CGram na posição 1
    }

    fun init() {
        Thread.sleep(20)

        writeCMD(48)
        Thread.sleep(5)

        writeCMD(48)
        Thread.sleep(1)

        writeCMD(48)
        writeCMD(56)
        writeCMD(8)
        writeCMD(1)
        writeCMD(6)
        writeCMD(15)
    }
}

/*Código feito ao som de:
    Snowfall by Oneheart
 */
```

F. Código Kotlin *SerialEmitter*

```
object SerialEmitter {
    private const val LCD_MASK = 0x01 //É o bit 0 (0001)
    private const val SSC_MASK = 0x02 //É o bit 1 (0010)
    private const val SDX_MASK = 0x08 //É o bit 3 (1000)
    private const val CLK_REG_MASK = 0x10 //É o bit 4 (0001 0000)
    private const val CLEAR_MASK = 0x1F // É os bits todos antes do 32(5) ou seja (0001 1111)

    // Envia tramas para os diferentes módulos Serial Receiver.
    enum class Destination { LCD, SCORE }

    // Inicia a classe
    fun init() {
        HAL.clrBits(CLEAR_MASK)
        HAL.setBits(LCD_MASK)
        HAL.setBits(SSC_MASK)
    }

    //  if LCD size = 10
    //  if Score size = 8

    // Envia uma trama para o SerialReceiver identificado o destino em addr,os bits de dados em ‘data’
e em size o número de bits a enviar.
    // Vai primeiro o cmd e depois a data
    fun send(addr: Destination, data: Int, size: Int) {
        var dataShifted = data
        var ParityBit = 0
        var counter = 0

        if (addr == Destination.LCD) {
            HAL.clrBits(LCD_MASK) //Entra negado no hardware
        } else if (addr == Destination.SCORE) {
            HAL.clrBits(SSC_MASK) //Entra negado no hardware
        }


        dataShifted = data.shl(3) //Mover a data 3 vezes pois enviamos pelo bit do SDX
        for (i in 0 until size - 1) {
            HAL.clrBits(CLK_REG_MASK) //Efetuar o clock
            HAL.writeBits(SDX_MASK, dataShifted) //Escrever o valor de dataShifted que está no valor de
SDX_MASK
            if (dataShifted.shr(3) % 2 == 1) {
                counter++
            }
            dataShifted = dataShifted.shr(1) //Mover a data um bit para escrevermos o bit seguinte
            HAL.setBits(CLK_REG_MASK) //Finalizar o clock
        }

        if (counter % 2 == 1) {
            ParityBit = 1 //Se o resto de divisão do counter
        }
        HAL.clrBits(CLK_REG_MASK) //Iniciar o clock
        HAL.writeBits(SDX_MASK, ParityBit.shl(3)) //Escrever o bit de paridade e o shift é para estar
em SDX
        HAL.setBits(CLK_REG_MASK) //Finalizar o clock
        HAL.setBits(LCD_MASK) //Colocar o bit do LCD
        HAL.setBits(SSC_MASK) //Colocar o bit do SSC
    }
}

/*Código feito ao som de:
    HIGHEST IN THE ROOM by Travis Scott
 */
```

G. Código Kotlin *ScoreDisplay*

```
object ScoreDisplay { // Controla o mostrador de pontuação.
    private const val OFF_MASK = 0xF //111 cmd 0001 data = 0001 111
    private const val ON_MASK = 0x7 // 111 cmd 0000 data = 0000 111
    private const val UPDATE_MASK = 0x6 // 110 cmd 0000 data = 0000 110
    private const val EMPTY_MASK = 0X78 // 1111 data 000 cmd depois na função somamos + cmd
    private var lastAnimationTime = System.currentTimeMillis() //Tempo da última animação
    private val animationInterval = 500L // 500 milliseconds
    private var animationCounter = 0 //Qual a última animação
    private val segments = arrayOf(0xA, 0xB, 0xC, 0xD, 0xE, 0x1) //lista das animações
    var isAnimating = true // animação do início é true inicialmente
    // Inicia a classe, estabelecendo os valores iniciais.
    fun init(){
        off(true)//desliga
        off(false)//liga
        for (cmd in 5 downTo 0){
            SerialEmitter.send(SerialEmitter.Destination.SCORE,cmd,8)   //enviar o cmd
        }
        SerialEmitter.send(SerialEmitter.Destination.SCORE, UPDATE_MASK, 8) //update para saber que
está ligado
    }

    // Envia comando para atualizar o valor do mostrador de pontuação
    fun setScore(value: Int){
        var value_temp=value //Para guardar o valor de entrada
        var divider = 100_000 //6 digitos de Score
        var leadingZero = true //0 à esquerda = true porque nunca começamos com número != 0 à esquerda
        for (cmd in 5 downTo 0) {
            var dividido = (value_temp / divider) * divider //Passar para um int e meter esse mesmo int
no resultado
            if (dividido == 0 && leadingZero && divider > 1) {
                SerialEmitter.send(SerialEmitter.Destination.SCORE, EMPTY_MASK + cmd, 8) //Se for 0 à
esquerda, apagar
            } else {
                SerialEmitter.send(SerialEmitter.Destination.SCORE, (value_temp / divider).shl(3) +
cmd, 8) //Escrever o valor
                leadingZero = false //Não é zero à esquerda
            }
            value_temp %= divider //Remover o número de maior grau
            divider /= 10 //Diminuir o grau do divisor
        }
        SerialEmitter.send(SerialEmitter.Destination.SCORE, UPDATE_MASK, 8)//Para dar update ao score
display
    }

    // Envia comando para desativar/ativar a visualização do mostrador de pontuação
    fun off(value: Boolean){
        if(value){
            SerialEmitter.send(SerialEmitter.Destination.SCORE, OFF_MASK, 8)//ligar o off

        }
        else SerialEmitter.send(SerialEmitter.Destination.SCORE, ON_MASK, 8)//não ligar o off
    }

    //Começa a animar novamente
    fun startAnimation() {
        isAnimating = true
    }

    //Para de animar
    fun stopAnimation() {
        isAnimating = false
    }
```

```
    // Muda a animação em questão
    fun updateAnimation() {
        if (isAnimating && System.currentTimeMillis() - lastAnimationTime >= animationInterval) {
            lastAnimationTime = System.currentTimeMillis() //Update ao lastAnimationTime
            animationCounter = (animationCounter + 1) % segments.size //O counter de animação = ao
resto de divisão pelo tamanho dos segmentos
            displayAlternatingShapes()
        }
    }

    //AlternatingShapes, bastante explicativo, muda as shapes (da animação inicial)
    private fun displayAlternatingShapes() {
        for (i in 0..5) {
            val cmd = (segments[animationCounter].shl(3)) + i //Shift 3 vezes porque é o cmd primeiro
            SerialEmitter.send(SerialEmitter.Destination.SCORE, cmd, 8) //Para definir qual dos
displays mandamos, daí serem em sincronia
        }
        SerialEmitter.send(SerialEmitter.Destination.SCORE, UPDATE_MASK, 8)//dar update a todos
    }
}

/*Código feito ao som de:
    Duvet by bôa
 */
```

H. Código Kotlin *TUI*

```
object TUI {
    //Escreve qualquer coisa, modo de manutenção feito à parte porque é usado mais vezes
    fun escrita(texto1: String, texto2: String) {
        LCD.clear()
        LCD.cursor(0, 0)
        for (i in texto1) {
            LCD.writeChar(i)
        }

        LCD.cursor(1, 0)
        for (i in texto2) {
            LCD.writeChar(i)
        }
    }

    //Limpa o LCD, somente existe para a ligação
    fun clear() {
        LCD.clear()
    }
    //Ligação...
    fun pressed_key(): Char = KBD.waitKey(10)

    //Se o valor na lista for > ele substitui por uma nave
    fun update() {
        LCD.cursor(0, 0)
        for (i in LCD1) {
            if (i == ">"){
                drawSHIP(0,1)
                continue
            }
            LCD.writeString(i)
        }

        LCD.cursor(1, 0)
        for (i in LCD2) {
            if (i == ">"){
                drawSHIP(1,1)
                continue
            }
            LCD.writeString(i)
        }
    }

    //Define a posição do cursor no LCD
    fun cursor(line:Int,column:Int){
        LCD.cursor(line,column)
    }

    //Escrever só um Char no LCD
    fun writeChar(c:Char){
        LCD.writeChar(c)
    }

    //Função que desenha a nave no jogo
    fun drawSHIP(line: Int, column: Int){
        LCD.drawShip(line, column)
    }
    //Função que desenha os aliens no jogo
    fun drawAlien(line:Int, column:Int){
        LCD.drawInvader(line,column)
    }

    //Funçã que liga e desliga o TUI
```

```
    fun off(mask:Boolean){
        if (mask){
            LCD.off(mask)
        }
        else {
            LCD.off(mask)
        }
    }
    fun init(){
        KBD.init()
        println("Load KBD completed")
        LCD.init()
        println("Load LCD completed")
        ScoreDisplay.off(false)
        LCD.off(false)
    }
}

fun main(){
    LCD.init()
    TUI.drawSHIP(1,1)
    TUI.drawAlien(1, 5)
}
/* Código feito ao som de:
    From The Start by Laufey
 */
```

I. Código Kotlin *M*

```
const val M_MASK = 0b10000000

object M {
    fun active(): Boolean{
        return HAL.isBit(M_MASK)
    }
}

fun main() {
    HAL.init()
    println(M.active())
    HAL.writeBits(M_MASK, 0xFF)
    println(M.active())
}
/*Código feito ao som de:
    Wannabe by Spice Girls
 */
```

J. Código Kotlin *CoinAcceptor*

```
object Coin_Acceptor{
    const val COIN_MASK = 0b0100_0000
    var last_state = false
    fun add(coins:Int):Int{
        var coinsAvailable=coins
        if (HAL.isBit(COIN_MASK) && (last_state==false)){
            coinsAvailable+=2 //Meter 2 coins na máquina
            last_state = true //Último estado é '1'
        } else if (!HAL.isBit(COIN_MASK) && last_state){
            HAL.clrBits(COIN_MASK) //Passa a máscara a 0 quando o valor não for bit(switch a 0)
            last_state=false //Último estado é '0'
        }
        return coinsAvailable
    }

    fun coinsActive():Boolean{
       return HAL.isBit(COIN_MASK)
    }
}
fun main(){
    HAL.init()
    Coin_Acceptor.add(2)
}
/*Código feito ao som de:
    Timber by Pitbull
 */
```

K. Código Kotlin *FileAccess*

```
import java.io.*

object FileAccess{

    //Efetua a leitura do ficheiro, separando a cada nova linha
    fun readFile(file: String): List<String> {
        val br = BufferedReader(FileReader(file))
        var line = br.readLine()
        val a = emptyList<String>().toMutableList()
        while (line != null) {
            val list = line.split("\n")
            a += list

            line = br.readLine()
        }
        br.close()
        return a
    }

    //Efetuar a atualização do ficheiro de acordo com os dados
    fun updateFile(file:String, list: List<String>){
        val pw = PrintWriter(file)
        for(i in list.indices){
            pw.println(list[i])
        }
        pw.close()
    }
}
/*Código feito ao som de:
    Not Like Us by Kendrick Lamar
 */
```

L. Código Kotlin *Scores*

```
import java.io.File

object Scores {
    const val LEADERBOARD_SIZE = 20 //Tamanho da leaderboard
    val alphabet = ('A'..'Z').toList() //Alfabeto numa lista
    val board = mutableListOf<Pair<Int, String>>() //Board, algo que mete int e string num par, para
podermos indexar de acordo com o int.
    var playerName = "" //Nome do player que começa como ""

    fun loadScores(filename: String) {
        board.clear() //Apaga o board todo
        File(filename).forEachLine { line ->
            val parts = line.split(";") //Agarra nas linhas e dá split quando tem ";"
            if (parts.size == 2) {
                val number = parts[0].trim().toIntOrNull() //A parte da esquerda do é o score do
jogador
                val name = parts[1].trim() //A parte direita é o nome do jogador
                if (number != null && name.isNotEmpty()) {
                    board.add(Pair(number, name)) //por fim se o valor de pontos não for nulo e o nome
não for vazio adiciona ao board
                }
            }

        }
        board.sortByDescending { it.first } //Organiza o board segundo o score do maior para o menor
    }
    //Guarda os resultados
    fun saveScores(filename: String) {
        File(filename).printWriter().use { writer ->
            board.take(LEADERBOARD_SIZE).forEach { (score, name) ->  //Agarra no board e para os 20
primeiros escreve o par de score ; name
                writer.println("$score;$name")
            }
        }
    }

    //Binary Search pelas kills(score) para determinar o indice em que é adicionado o valor
    fun findInsertionIndex(newScore: Int): Int {
        var low = 0
        var high = board.size - 1
        while (low <= high) {
            val mid = (low + high) / 2
            when {
                board[mid].first < newScore -> high = mid - 1
                board[mid].first > newScore -> low = mid + 1
                else -> return mid
            }
        }
        return low
    }

    //função que atualiza a leaderboard, juntanto todas as anteriores
    fun updateLeaderboard() {
        if (!M.active()){
        val index = findInsertionIndex(kills) //descobre o index
        val playerScore = Pair(kills, playerName) //Associa as kills a um nome
        if(index >= board.size){
            board += playerScore //Index muito grande(maior do que o board) só adicionamos
        }
        if (index < LEADERBOARD_SIZE && index < board.size) { //Se o index for menor que o tamanho da
leaderboard e do que o board
```

```
            board.add(index, playerScore) //Adicionamos
            if (board.size > LEADERBOARD_SIZE) {
                board.removeAt(LEADERBOARD_SIZE) //Removemos os que são maiores que o tamanho da
leaderboard
            }
        }
        saveScores("SIG_Scores.txt") //Guardamos os scores
    }
    }
}
/*Código feito ao som de:
       Dreams by RUBII
*/
```

M. Código Kotlin *Statistics*

```
var games = 0
var total_coins = 0
object Statistics{
    //Efetuar a leitura do ficheiro e associar o que está na primeira linha a games e a segunda a
total_coins
    fun readAndKeep(){
        val stats = FileAccess.readFile("Statistics.txt")

        games = stats[0].toInt()
        total_coins = stats[1].toInt()

    }

    //Atualizar o ficheiro para colocar os dados de games e coins a 0
    fun eraseStats(){
        FileAccess.updateFile("Statistics.txt", listOf(0.toString(), 0.toString()))
    }

    //Guardar no ficheiro o valor de games e de coins atual
    fun saveToFile(games: Int, coins: Int) {
        FileAccess.updateFile("Statistics.txt", listOf(games.toString(), coins.toString()))
    }
}
/*Código feito ao som de:
    STARGAZING by Travis Scott
 */
```

N. Código Kotlin *App*

```
import isel.leic.utils.Time
import kotlin.system.exitProcess

const val alternancyRate = 3000 //Mudança de menu no menu inicial a cada 3 segundos
const val spawnRate = 1000 // Aparece um novo inimigo a cada segundo

var LCD1 = mutableListOf(" ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ")
var LCD2 = mutableListOf(" ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ")
val NUMBERS = listOf("0","1","2","3","4","5","6","7","8","9")

var gameover = false
var acertou = false
var start = false

var countIteration = 0
var counterMenu = 0
var linha = 0
var coins = 0
var coinsTemp = 0
var kills = 0

fun main(){
    TUI.init()
    ScoreDisplay.init()
    Statistics.readAndKeep() // Ler as estatisticas
    Scores.loadScores("SIG_Scores.txt") // Ler o leaderboard
    while (true){
        mainLoop()
        resetGame()
    }
}
//Loop principal (jogo acontece aqui)
fun mainLoop(){

    var lastTime = System.currentTimeMillis()
    if (!M.active())
        updateCoinsDisplay(coins) //Escreve o valor de coins a $0
```

```
menu()
ScoreDisplay.setScore(0) //Começa o jogo com Score a 0
kills = 0
gameover = false
start = true
ScoreDisplay.stopAnimation() //Para as animações
while (start){
    ScoreDisplay.updateAnimation() //Dar update ao valor do Score Display
    val currentKey = TUI.pressed_key().toString()

    if (currentKey != KBD.NONE.toString() && currentKey != "*" && currentKey != "#") {
        // Dar load ao bacano
        if (linha%2 == 0) {
            LCD1[0] = currentKey
        }
        if (linha%2 == 1) {
            LCD2[0] = currentKey
        }
        TUI.update()
    }

    if (currentKey == "#") {
        // Disparar
        eliminateEnemy()
    }

    switchLine(currentKey)

    val currentTime = System.currentTimeMillis()
    if (currentTime - lastTime >= spawnRate) {
        spawn()
        lastTime = currentTime //Atualizar o valor para voltar a esperar 1 segundo
    }
    if(gameover){
        gameOver()
        start=false
    }
}
```

```
    resetGame()
}
//Função que efetua a mudança de linha do LCD ao pressionarmos '*' enquanto estivermos no jogo
fun switchLine (key : String){
    if (key == "*") {
        linha = ++linha
        if(linha%2 == 0){
            LCD1[1] = ">"
            LCD2[1] = " "
            LCD1[0] = "]"
            LCD2[0] = "]"
        }
        if(linha%2 == 1){
            LCD2[1] = ">"
            LCD1[1] = " "
            LCD2[0] = "]"
            LCD1[0] = "]"
        }
        TUI.update() //Para efetuar a substituição de ">" pela nave
    }
}

//Função que converte de string para Int
fun convert(string : String) : Int {
    return string.toInt()
}

fun eliminateEnemy(){
    if((firstElementEmpty(LCD1)+1) in 0..15){
    val enemyOfLine1 = LCD1[lastElementFull(LCD1)]//VALOR DO ALVO


    // Caso da primeira linha
    if (LCD1[0] == LCD1[firstElementEmpty(LCD1)+1] && firstElementEmpty(LCD1) <= 15){
        LCD1[firstElementEmpty(LCD1)+1] = " " //
        LCD1[0] = "]" //
        kills += (convert(enemyOfLine1) + 1) //
        ScoreDisplay.setScore(kills) //Meter o scoreDisplay com o valor de kills
```

```
        acertou=true
    }
    }
    if((firstElementEmpty(LCD2)+1) in 0..15 ){
    val enemyOfLine2 = LCD2[lastElementFull(LCD2)] //

    // Caso da segunda linha
    if (LCD2[0] == LCD2[firstElementEmpty(LCD2)+1] && firstElementEmpty(LCD2) <= 15){
        LCD2[firstElementEmpty(LCD2)+1] = " " //
        LCD2[0] = "]" //
        kills += (convert(enemyOfLine2) + 1) //
        ScoreDisplay.setScore(kills)        //Meter o scoreDisplay com o valor de kills
        acertou=true
    }
    }
    TUI.update()                            //Dar update ao TUI para termos o número a desaparecer
}

fun spawn(){
    val enemy = NUMBERS.random()                        // Gerar um número aleatório para ser o próximo
inimigo
    val enemyLine = (0..1).random()             // Gerar um número entre 0 e 1 para decidir a linha onde
estará o próximo inimigo

    if(firstElementEmpty(LCD1) <= 1 || firstElementEmpty(LCD2) <= 1){ //!Condição de derrota!
        gameover = true
        return
    }
    if (enemyLine==0){ //casos do spawn ser na linha 1
        for (i in firstElementEmpty(LCD1)..<LCD1.size-1){ //como é downto
            LCD1[i] = LCD1[i+1] //i vai para i+1(-1 no LCD)
            LCD1[i+1] = " " //E o i+1 vai ser espaço, libertando o i de número 15
        }
        LCD1[LCD1.size-1] = enemy //Para colocar o inimigo no mesmo (o mais à direita)
        TUI.update()
    }
    else if (enemyLine==1){ //casos do spawn ser na linha 2
        for (i in firstElementEmpty(LCD2)..<LCD2.size-1){ //como é downto
```

```
            LCD2[i] = LCD2[i+1] //i vai para i+1(-1 no LCD)
            LCD2[i+1] = " " //E o i+1 vai ser espaço, libertando o i de número 15
        }
        LCD2[LCD2.size-1] = enemy //Para colocar o inimigo no mesmo (o mais à direita)
        TUI.update()
    }
}

// Retorna o indice do elemento vazio ao lado de um inimigo
fun firstElementEmpty(list: MutableList<String>):Int {
    for (i in list.size - 1 downTo 0) {
        if (list[i] == " ") {
            return i
        }
    }
    return -1
}

// Retorna o indice do elemento da ponta esquerda
fun lastElementFull(list: MutableList<String>):Int{
    for (i in 0..<list.size) {
        if (list[i] in NUMBERS) {
            return i
        }
    }
    return -1
}

fun gameOver(){         // Função que dá handle ao estado de Game Over
    start=false
    val timeinit = Time.getTimeInMillis()

    var time:Long = System.currentTimeMillis()
    var blinkscore = true

    val updateInterval = 250 // Update interval in milliseconds
    var lastUpdateTime = timeinit
    while (gameover){
```

```
        while (time - timeinit <= 3000) {
            time = System.currentTimeMillis()
            if (time - lastUpdateTime >= updateInterval) {
                if (blinkscore) {
                    ScoreDisplay.off(true)//Desliga o score display
                    TUI.escrita(" "," ") //"Desliga o LCD"
                } else {
                    TUI.escrita("   GAME  OVER   ", "   Score:$kills") //Escreve o texto de final
                    ScoreDisplay.off(false) //Liga o score display
                    ScoreDisplay.setScore(kills) //Com o valor de kills
                }
                blinkscore = !blinkscore //altera o valor da variavel blinkscore para o inverso
                lastUpdateTime = time //Atauliza o valor de lastUpdateTime
            }
        }
        while (Time.getTimeInMillis()-timeinit >= 3000) {
            if (M.active()) { //Se estivermos em manutenção não temos de dar input a nome
                gameover=false
                LCD1 = mutableListOf(" ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", "
", " ", " ")
                LCD2 = mutableListOf(" ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", "
", " ", " ")
                TUI.update()
                TUI.clear()
                kills=0
                ScoreDisplay.setScore(kills)
                mainLoop()
                menu()
            } else { //Caso contrário devemos fazê-lo
                gameover = false
                inputPlayerName()
                kills = 0
                ScoreDisplay.setScore(kills)
                break
            }
        }
    }
}
```

```
fun menu() {
    var update = false
    var kTemp: String
    var lastTime= System.currentTimeMillis()
    ScoreDisplay.startAnimation() //Começa a animação do Score Display
    while (!start) {
        ScoreDisplay.updateAnimation() //Atualiza a animação do Score Display
        kTemp = TUI.pressed_key().toString()

        if (!M.active()) { //Se não estivermos em manutenção
            if (update) { //E update for true(sairmos de manutenção)
                updateCoinsDisplay(coins) //Escrevemos o menu com o valor de coins atual
                update = false //update a falso pois o último estado não é manutenção
            }

            val currentTime = System.currentTimeMillis()
            if (currentTime - lastTime >= alternancyRate) { //A cada 3 segundos
                switchMenu() //Alterar entre o menu de score ou de coins
                lastTime = currentTime
            }

            coins = Coin_Acceptor.add(coins) //Se o M não estiver ativo aumentamos os coins ao mover o
switch a '1'
            if (coinsTemp != coins) { //coinsTemp diferente dos coins antigos pois somámos 2 (garantir
que tudo correu bem)
                total_coins++ //Aumentar o total de moedas no ficheiro de statistics em 1
                updateCoinsDisplay(coins) //Reescrever o menu de coins com o novo valor
            }
            coinsTemp = coins //Atualizar os coinsTemp

            if (kTemp == "*" && coins > 0 && !Coin_Acceptor.coinsActive()) {
                initGame(kTemp) //Se o valor de coins for maior que 0 e clicarmos no "*" o jogo inicia
            }
        }

        if (M.active()) { //Se o M estiver ativo
            update = true //Ativamos a flag de update
            TUI.escrita(" On Maintenance ", " *-Count #-ShutD") //Escrevemos o menu de manutenção
```

```
            handleMaintenanceMode() //Vamos para a função que manuseia a manutenção
        }
    }
}

fun initGame(kTemp: String){
    switchLine(kTemp)// Desenhar a nave
    switchLine(kTemp)// Voltar à linha de cima
    games++ //Aumentar o número de jogos no ficheiro de statistics
    coins-- //Diminuir em 1 o valor de coins atuais na máquina
    start = true //Colocar a flag de início a 1 para garantir que começou
}

fun showStatistics() {
    TUI.escrita("Games:$games", "Coins:${total_coins}") //Mostrar os valores que estão no ficheiro
Statistics
    val timeinit = Time.getTimeInMillis()
    var time : Long = 0
    while (time - timeinit <= 5000) {
        time = Time.getTimeInMillis()
        if (!M.active()){ //Se o M for inativo volta ao Menu
            menu()
        }
        val teclado = TUI.pressed_key()
        when (teclado) {
            '#' -> eraseStats() //Se pressionarmos a tecla "#" colocamos as estatísticas a 0
            ' ' -> Unit
            else -> { //Caso apertemos outra tecla, verificamos se o M estiver ativo e damos break para
voltarmos ao mainLoop
                M.active()
                break
            }
        }
    }
    mainLoop() //Ir ao mainLoop que nos vai levar ao menu
}

//Função que escreve o texto do menu inicial de acordo com os coins existentes na máquina
```

```
fun updateCoinsDisplay(coins: Int) {
    val cointext = "$$coins"
    TUI.escrita(" Space Invaders", " Game           ".dropLast(cointext.length) + cointext)
    TUI.drawSHIP(1,6)
    TUI.drawAlien(1,9)
    TUI.drawAlien(1,11)
}

fun handleMaintenanceMode() {
    while (M.active()) {
        if (Coin_Acceptor.add(coins) == coins+2){ //Impedir que os coins sejam aumentados quando estamos
em Manutenção
            coins = coins
        }
        val tecla = TUI.pressed_key()
        when (tecla) {
            '*' -> showStatistics()
            '#' -> handleShutdown()
            ' ' -> Unit
            else -> {
                switchLine("*") // Desenhar a nave
                switchLine("*") // Voltar à linha de cima
                start = true
                mainLoop()
            }
        }
    }
}

fun handleShutdown() {
    TUI.escrita("    Shutdown   ", "5-Yes Other - No") //Escrita do menu de Shutdown
    val timeinit = Time.getTimeInMillis()
    var time:Long = 0
    while (time-timeinit<=5000) {
        time=Time.getTimeInMillis()
        if (!M.active()){ //Se o M for inativo voltar ao menu
            menu()
        }
```

```
        val decision = TUI.pressed_key()
        when (decision) {
            '5' -> {
                Statistics.saveToFile(games, total_coins) //Se clicarmos no 5 desligamos a máquina e
guardamos os jogos e coins inseridos no Statistics
                TUI.off(true)
                ScoreDisplay.off(true)
                exitProcess(0)
            }
            ' ', '#' -> Unit //Não fazemos nada
            else -> mainLoop() //Vamos ao mainLoop entrar no menu
        }
    }
    mainLoop()
}

fun eraseStats() {
    TUI.escrita("Clear Counters?", " 5-Yes Other-No") //Escrita do menu de clear
    val timeinit = Time.getTimeInMillis()
    var time:Long = 0

    while (time-timeinit<=5000) {
        val key = TUI.pressed_key()
        when (key) {
            '5' -> {
                Statistics.eraseStats() //Se clicar-mos no 5 apagamos as Estatísticas
                Statistics.readAndKeep() //E efetuamos a leitura para termos os counters a 0,0
                menu() //Voltamos ao menu para voltar ao menu de manutenção
            }
            ' ' -> Unit
            else -> menu()
        }
        time=Time.getTimeInMillis()
    }
    mainLoop()
}
```

```
fun formatScoresCorrectly():String{ //Função que retorna a quantidade de espaços correta de acordo com o tamanho do nome e do número de kills
    var numberOfSpaces = 16
    numberOfSpaces = numberOfSpaces - "${counterMenu+1}-${Scores.board[counterMenu].second}".length - "${Scores.board[counterMenu].first}".length
    return " ".repeat(numberOfSpaces)
}
fun switchMenu() {//Função que altera o menu que está a ser apresentado
    if (countIteration % 2 == 0){ //Caso estejamos numa iteração par, mostramos o número de coins
        updateCoinsDisplay(coins)
    }
    else { //Caso contrário, demonstramos o valor do counter + 1 pois o counter é iniciado a 0, seguido do nome e por fim o score
        TUI.escrita("   Space   Invaders   ",   "${counterMenu+1}-${Scores.board[counterMenu].second}" +formatScoresCorrectly() + "${Scores.board[counterMenu].first}")
        counterMenu++ //Após a escrita aumentamos o counter para a próxima iteração
    }
    countIteration++ //Aumentamos a iteração para mudarmos entre ser par ou ímpar
    if (counterMenu >= Scores.board.size) counterMenu = 0 //Se o valor do counter for superior ao tamanho de board ele regressa a 0
}
fun inputPlayerName() { //Função que manuseia a forma como o nome do jogador é implementado

    LCD1 = mutableListOf(" ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ")
    TUI.escrita("Name:", "Score:$kills") //Escrevemos o menu inicial iniciando o cursor na coluna seguinte

    var column = 5
    var currentLetterIndex = 0
    TUI.cursor(0, column)

    while (true) {
        val key = TUI.pressed_key()
        when (key) {
            '8' -> if (currentLetterIndex > 0) currentLetterIndex-- //Diminui o index da letra na lista do alfabeto caso seja maior que 0
            '2' -> if (currentLetterIndex < Scores.alphabet.size - 1) currentLetterIndex++ //Aumenta o index da letra caso esta seja menor que o máximo
```

```
            '6' -> if (column < 12) column++ //Aumenta a coluna onde estamos a escrever
            '4' -> if (column > 5) column-- //Diminui a coluna onde estamos a escrever
            '5' -> {
                Scores.playerName = LCD1.subList(5, LCD1.indexOfLast { it != " " } + 1).joinToString("")
                break
            }
        }
        TUI.writeChar(Scores.alphabet[currentLetterIndex]) //Escrevemos o Char correspondente ao index
atual
        TUI.cursor(0, column)
        LCD1[column] = Scores.alphabet[currentLetterIndex].toString()

    }
    Scores.updateLeaderboard()
}

fun resetGame(){
    TUI.clear()
    gameover = false
    linha = 0
    LCD1 = mutableListOf(" ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ")
    LCD2 = mutableListOf(" ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ")
    start = false
}
/*Código feito a ouvir:
    Katy Perry no seu prime
 */
```