ISEL
Departamento de Engenharia Eletrónica e Telecomunicações e de Computadores

Licenciatura em Engenharia Informática e de Computadores

Jogo Invasores Espaciais (*Space Invaders Game*)

Afonso Leote, 51700

Luka Roca, 51820

Diogo Leitão, 51634

Projeto
de
Laboratório de Informática e Computadores
2023 / 2024
verão

12 de junho de 2024

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; 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.

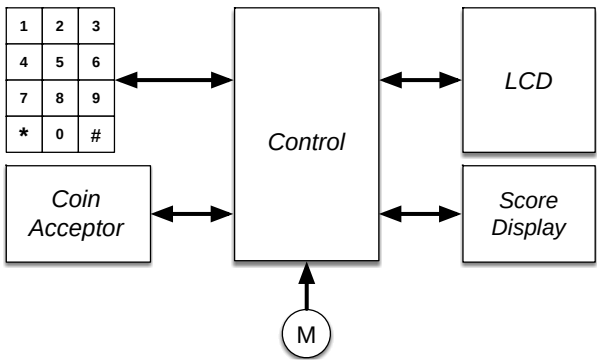

Figura 1 – Diagrama de blocos do jogo Invasores Espaciais (*Space Invaders Game*)

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 10 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.
- **Iniciar 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*.

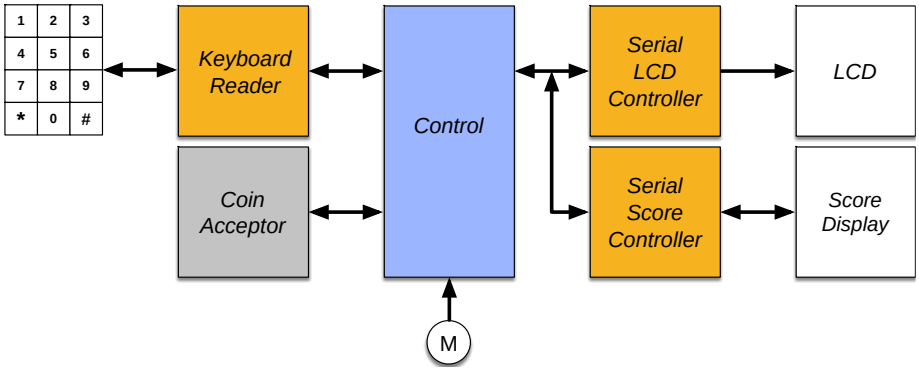

Figura 2 – Arquitetura do sistema que implementa o jogo Invasores Espaciais (*Space Invaders Game*)

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 3.

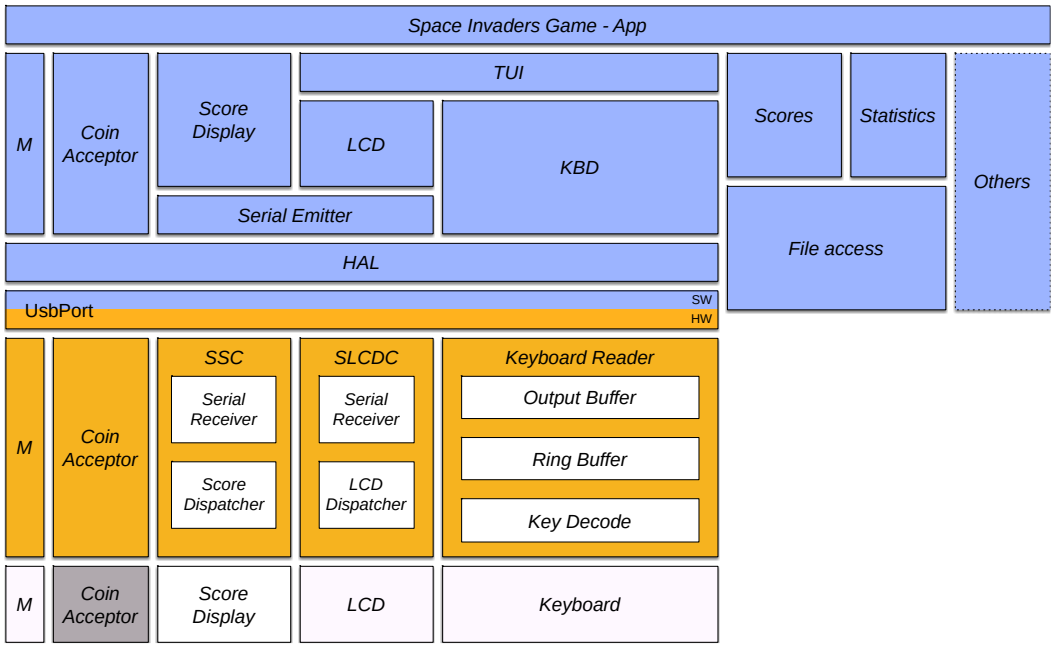

Figura 3 – Diagrama lógico do Jogo Invasores Espaciais (*Space Invaders Game*)

3 Implementação do sistema

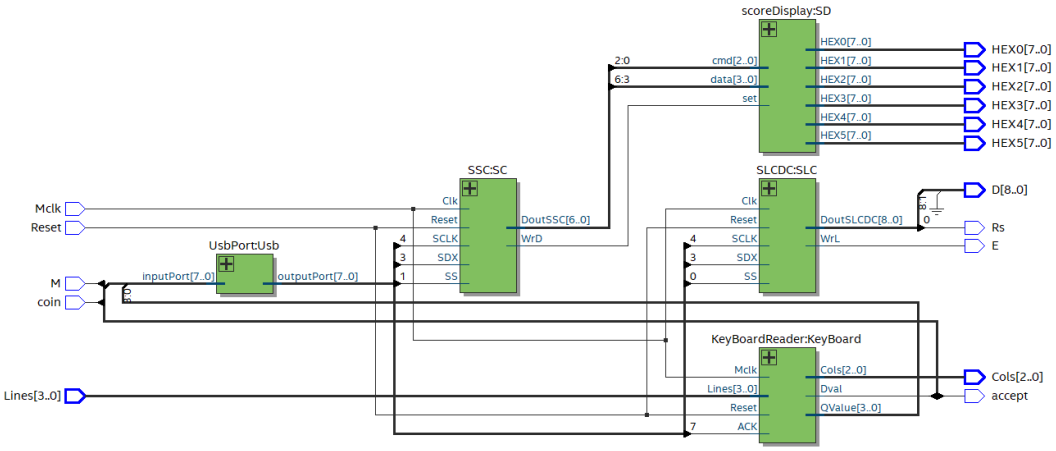

Figura 4 – Diagrama de blocos do SpaceInvadersGame

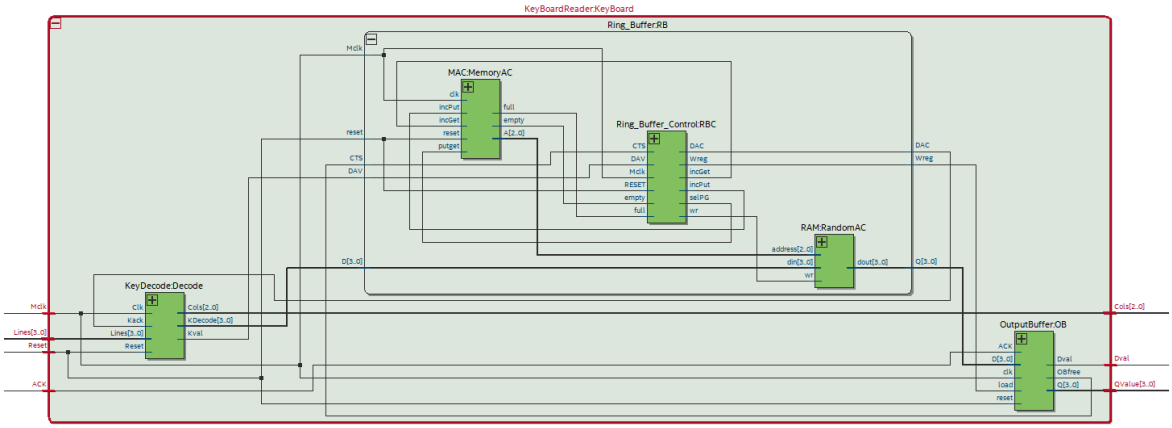

Figura 5 – Diagrama de blocos do KeyBoardReader

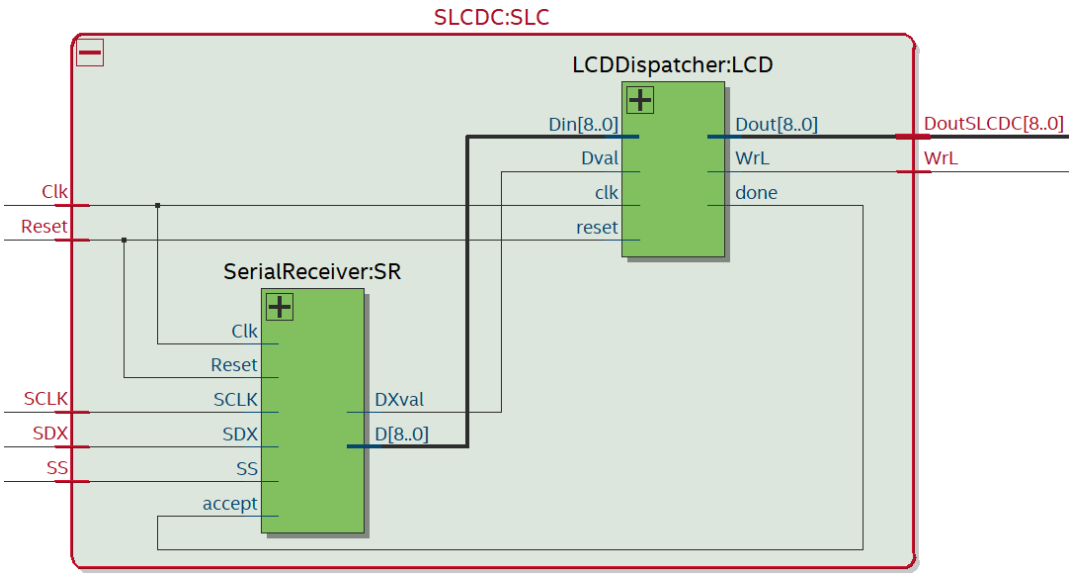

Figura 6 – Diagrama de blocos do SLCDC

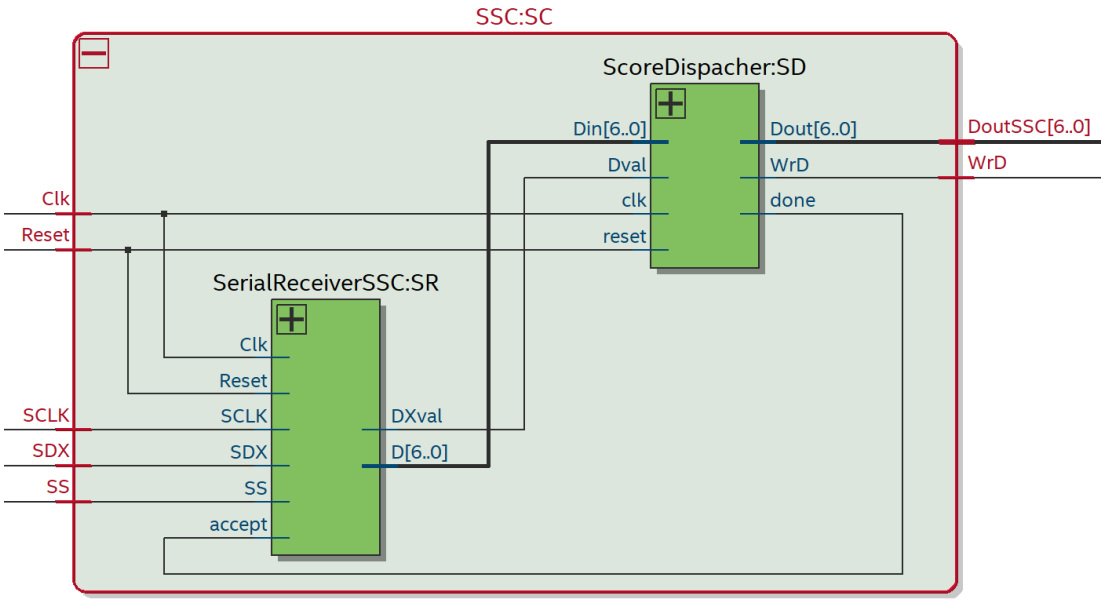

Figura 7 – Diagrama de blocos do SSC

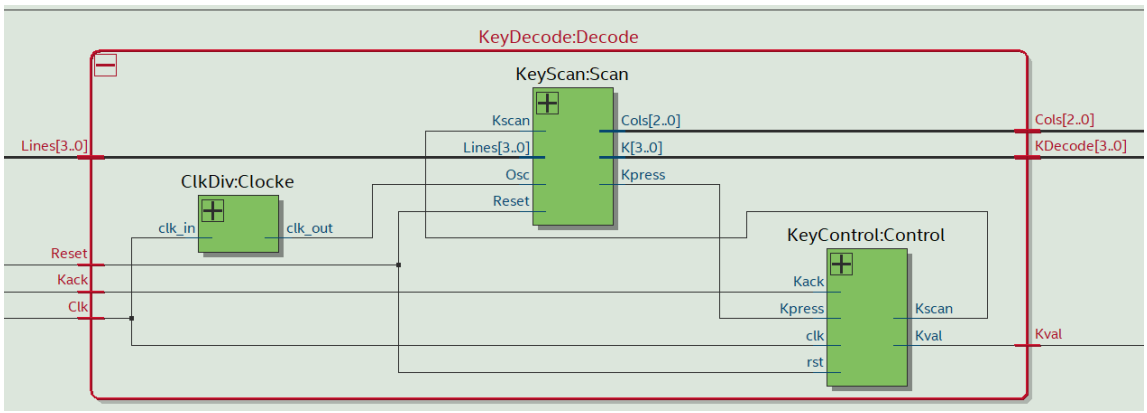

Figura 8 – Diagrama de blocos do KeyDecode

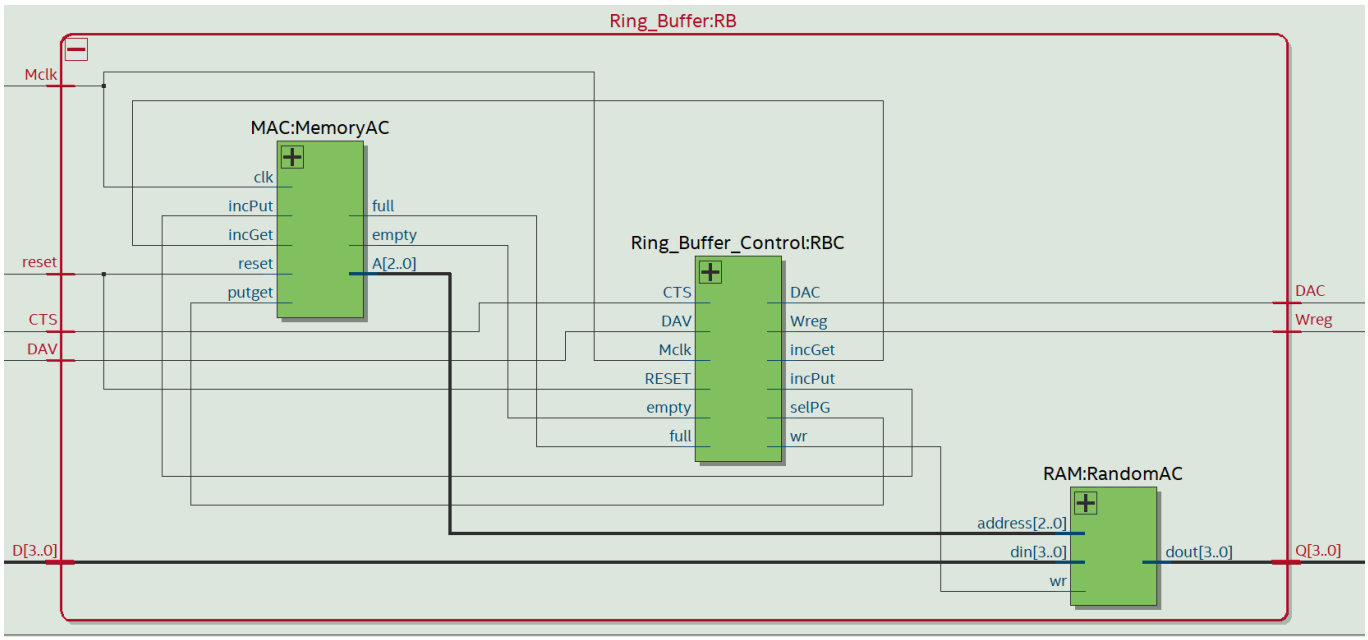

Figura 9 – Diagrama de blocos do Ring_Buffer

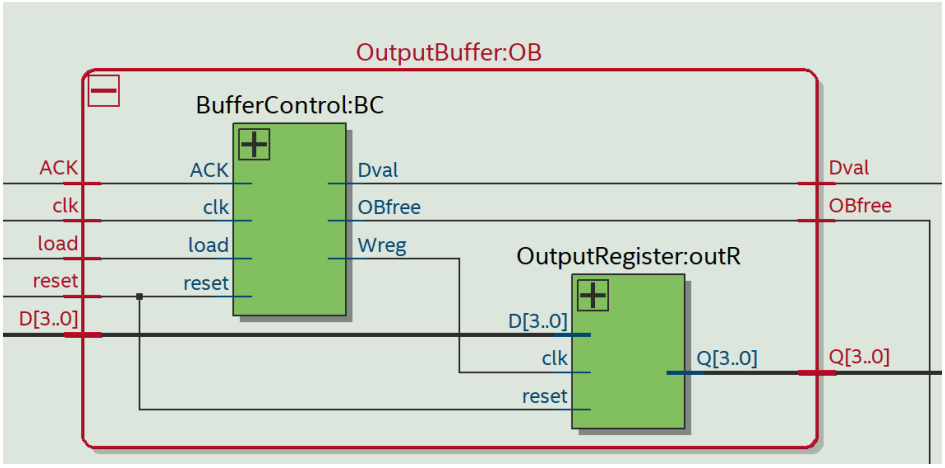

Figura 10 – Diagrama de blocos do OutputBuffer

4 Conclusões

O projeto foi concluído com sucesso, atingindo todos os objetivos propostos.

Durante a fase de desenvolvimento, todo o hardware foi realizado em VHDL enquanto que o software foi desenvolvido em Kotlin, tendo sido estas duas unidades testadas separadamente e validadas, na placa e na simulação, respetivamente.

Na fase final do projeto estas duas unidades foram testadas em conjunto por meio de utilização de um componente denominado por UsbPort que permitiu a interligação entre o hardware e o software.

Por fim, o projeto foi validado por completo na placa DE10-Lite e foi possível observar o seu correto funcionamento, confirmando assim que os objetivos pretendidos foram alcançados com sucesso.

A. Interligações entre o HW e SW

Durante o desenvolvimento do projeto, foi necessário interligar o hardware criado no Quartus em VHDL e o software desenvolvido em Kotlin. A solução envolve a utilização de um componente chamado UsbPort, composto por oito entradas e oito saídas, embora ao longo do projeto tenham sido usadas apenas cinco entradas e cinco saídas.

Por exemplo, quando uma tecla é pressionada no teclado matricial 4x4, essa tecla é detetada pelo Keyboard Reader em hardware, e esse valor é colocado nas primeiras quatro entradas do UsbPort para que o software descodifique o binário lido no UsbPort e, se apropriado, exiba o mesmo valor no LCD. Na figura 4, são mostradas as conexões do UsbPort com os outros componentes.

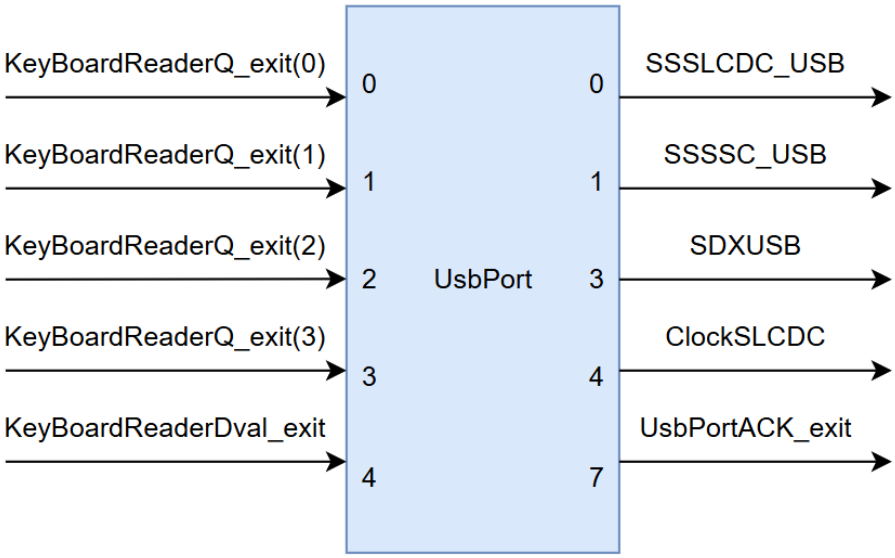

Figura 4- Entradas e saídas do UsbPort

B. Atribuição de Pinos

```
set_location_assignment PIN_W8     -to Rs
set_location_assignment PIN_V5     -to E

#Pins LCD
set_location_assignment PIN_AA15   -to D[0]
set_location_assignment PIN_W13    -to D[1]
set_location_assignment PIN_AB13   -to D[2]
set_location_assignment PIN_Y11    -to D[3]
set_location_assignment PIN_W11    -to D[4]
set_location_assignment PIN_AA10   -to D[5]
set_location_assignment PIN_Y8     -to D[6]
set_location_assignment PIN_Y7     -to D[7]
set_location_assignment PIN_Y6     -to D[8]

#Entradas

set_location_assignment PIN_P11    -to Mclk

set_location_assignment PIN_D12    -to Reset #Switch Ldr2-> 3

set_location_assignment PIN_C12    -to coin #Switch Ldr3-> 4

set_location_assignment PIN_A12    -to M #Switch Ldr4-> 5

#Pins Linhas
set_location_assignment PIN_W5     -to  Lines[0]
set_location_assignment PIN_AA14   -to      Lines[1]
set_location_assignment PIN_W12    -to  Lines[2]
set_location_assignment PIN_AB12   -to      Lines[3]

#Pins Colunas
set_location_assignment PIN_AB11   -to      Cols[0]
set_location_assignment PIN_AB10   -to      Cols[1]
set_location_assignment PIN_AA9    -to      Cols[2]

#Pins Score
set_location_assignment PIN_C14    -to       HEX0[0]
set_location_assignment PIN_E15    -to       HEX0[1]
set_location_assignment PIN_C15    -to       HEX0[2]
set_location_assignment PIN_C16    -to       HEX0[3]
set_location_assignment PIN_E16    -to       HEX0[4]
set_location_assignment PIN_D17    -to       HEX0[5]
set_location_assignment PIN_C17    -to       HEX0[6]
set_location_assignment PIN_D15    -to       HEX0[7]
set_location_assignment PIN_C18    -to       HEX1[0]
set_location_assignment PIN_D18    -to       HEX1[1]
set_location_assignment PIN_E18    -to       HEX1[2]
set_location_assignment PIN_B16    -to       HEX1[3]
set_location_assignment PIN_A17    -to       HEX1[4]
set_location_assignment PIN_A18    -to       HEX1[5]
set_location_assignment PIN_B17    -to       HEX1[6]
set_location_assignment PIN_A16    -to       HEX1[7]
set_location_assignment PIN_B20    -to       HEX2[0]
set_location_assignment PIN_A20    -to       HEX2[1]
set_location_assignment PIN_B19    -to       HEX2[2]
set_location_assignment PIN_A21    -to       HEX2[3]
set_location_assignment PIN_B21    -to       HEX2[4]
```

```
set_location_assignment PIN_C22    -to       HEX2[5]
set_location_assignment PIN_B22    -to       HEX2[6]
set_location_assignment PIN_A19    -to       HEX2[7]
set_location_assignment PIN_F21    -to       HEX3[0]
set_location_assignment PIN_E22    -to       HEX3[1]
set_location_assignment PIN_E21    -to       HEX3[2]
set_location_assignment PIN_C19    -to       HEX3[3]
set_location_assignment PIN_C20    -to       HEX3[4]
set_location_assignment PIN_D19    -to       HEX3[5]
set_location_assignment PIN_E17    -to       HEX3[6]
set_location_assignment PIN_D22    -to       HEX3[7]
set_location_assignment PIN_F18    -to       HEX4[0]
set_location_assignment PIN_E20    -to       HEX4[1]
set_location_assignment PIN_E19    -to       HEX4[2]
set_location_assignment PIN_J18    -to       HEX4[3]
set_location_assignment PIN_H19    -to       HEX4[4]
set_location_assignment PIN_F19    -to       HEX4[5]
set_location_assignment PIN_F20    -to       HEX4[6]
set_location_assignment PIN_F17    -to       HEX4[7]
set_location_assignment PIN_J20    -to       HEX5[0]
set_location_assignment PIN_K20    -to       HEX5[1]
set_location_assignment PIN_L18    -to       HEX5[2]
set_location_assignment PIN_N18    -to       HEX5[3]
set_location_assignment PIN_M20    -to       HEX5[4]
set_location_assignment PIN_N19    -to       HEX5[5]
set_location_assignment PIN_N20    -to       HEX5[6]
set_location_assignment PIN_L19    -to       HEX5[7]

#Pins Leds
#set_location_assignment PIN_A8     -to DXval
#set_location_assignment PIN_A9     -to D[0]
#set_location_assignment PIN_A10    -to D[1]
#set_location_assignment PIN_B10    -to D[2]
#set_location_assignment PIN_D13    -to D[3]
#set_location_assignment PIN_C13    -to D[4]
#set_location_assignment PIN_E14    -to D[5]
#set_location_assignment PIN_D14    -to D[6]
#set_location_assignment PIN_A11    -to D[7]
#set_location_assignment PIN_B11    -to D[8]
```

C. Código Kotlin *HAL*

```
import isel.leic.UsbPort
// Virtualiza o acesso ao sistema UsbPort
object HAL {
    var lastOutput = 0
    // Inicia a classe
    fun init () {
        lastOutput = 0
        UsbPort.write(lastOutput)
    }
    // Retorna true se o bit tiver o valor lógico '1'
    fun isBit (mask:Int) : Boolean {
        return UsbPort.read().and(mask) != 0
    }
    // 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) {
        clrBits(mask)
        setBits(value and mask)
    }
    // Coloca os bits representados por mask no valor lógico '1'
    fun setBits (mask: Int) {
        lastOutput = mask.or(lastOutput)
        UsbPort.write(lastOutput)
    }
    // Coloca os bits representados por mask no valor lógico '0'
    fun clrBits(mask:Int) {
        lastOutput = lastOutput.and(0xFF - mask)
        UsbPort.write(lastOutput)
    }
}
```

D. Código Kotlin *KBD*

```
import isel.leic.utils.Time
// Ler teclas. Métodos retornam '0'..'9','#','*' ou NONE.
object KBD {
    const val KEY = 0X0F
    const val NONE = ' '
    const val KACK_MASK = 0X80
    const val KVAL_MASK = 0X10
    val keys: CharArray = charArrayOf('1', '4', '7', '*', '2', '5', '8', '0', '3',
'6', '9', '#')

    fun init() {//Inicia a classe
        HAL.init()
    }
    //Retorna de imediato a tecla premida ou NONE se não há tecla premida.
    fun getKey(): Char {
        if (HAL.isBit(KVAL_MASK)) {
            val tecla = HAL.readBits(KEY)
            HAL.setBits(KACK_MASK)
            if(!HAL.isBit(KVAL_MASK)) {
                HAL.clrBits(KACK_MASK)
                return keys[tecla]
            }
        }
        return NONE
    }
    //Retorna a tecla premida, caso ocorra antes do 'timeout' (representado em
milissegundos), ou NONE caso contrário.
    fun waitKey(timeout: Long): Char {
        val timeFinal = timeout + Time.getTimeInMillis() //tempo atual em
milisegundos + timeout
        var key = NONE
        do {
            key = getKey()
        } while (Time.getTimeInMillis() < timeFinal && key == NONE)
        return key
    }
}
```

E. Código Kotlin *LCD*

```
import isel.leic.utils.Time

object LCD { // Escreve no LCD usando a interface a 4 bits.
    //    const val LINES = 2
    val COLS = 16// Dimensão do display.
    const val E_MASK = 0X20
    const val RS_MASK = 0x10
    const val CLK_MASK = 0X40
    const val DATA_MASK = 0X0F

    // Envia a sequência de iniciação para comunicação a 4 bits.
    fun init() {
        SerialEmitter.init()
        writeCMD(0b00110000)
        Time.sleep(15)
        writeCMD(0b00110000)
        Time.sleep(1)
        writeCMD(0b00110000)
        writeCMD(0b00111000)
        writeCMD(0b00001000)
        writeCMD(0b00000001)
        writeCMD(0b00000110)
        writeCMD(0b00001111)
        loadCustomChars()
    }

    fun writeByteSerial(rs: Boolean, data: Int) {
        val rS = if (rs) 1 else 0
        SerialEmitter.send(addr = SerialEmitter.Destination.LCD, data.shl(1) + rS,
9)
    }

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

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

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

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

    // Escreve uma string na posição corrente.
    fun write(text: String) {
        for (c in text) {
            write(c)
        }
```

```
    }

    // Envia comando para posicionar cursor ('line':0..LINES-1 , 'column':0..COLS-1)
    fun cursor(line: Int, column: Int) {
        writeCMD((line * 0X40 + column) or 0x80)
    }

    // Envia comando para limpar o ecrã e posicionar o cursor em (0,0)
    fun clear() {
        writeCMD(0b00000001)
        cursor(0, 0)
    }

    fun invader() {
        writeCMD (0x40)
        writeDATA(0x1F)
        writeDATA(0x1F)
        writeDATA(0x15)
        writeDATA(0x1F)
        writeDATA(0x1F)
        writeDATA(0x11)
        writeDATA(0x11)
        writeDATA(0x00)
    }

    fun spaceShip() {
        writeCMD (0x48)
        writeDATA(0x1E)
        writeDATA(0x18)
        writeDATA(0x1C)
        writeDATA(0x1F)
        writeDATA(0x1C)
        writeDATA(0x18)
        writeDATA(0x1E)
        writeDATA(0x00)
    }
    fun loadCustomChars() {
        invader()
        spaceShip()
    }
}
```

F. Código Kotlin *SerialEmitter*

```
object SerialEmitter { // Envia tramas para os diferentes módulos Serial Receiver.
    enum class Destination { LCD, SCORE }
    private const val SSLCD_MASK = 0x01
    private const val SSSCORE_MASK = 0x02
    private const val SCLK_MASK = 0x10
    private const val SDX_VAL = 0x08
    fun init() {
        HAL.init()
        HAL.setBits(SSLCD_MASK)
        HAL.setBits(SSSCORE_MASK)
        HAL.clrBits(SCLK_MASK)
    }
    fun send(addr: Destination, data: Int, size : Int){
        val adress = if(addr == Destination.LCD) SSLCD_MASK else SSSCORE_MASK
        var parity = 0
        var paridadeFinal = 0x01
        var count= 0
        HAL.clrBits(adress)
        repeat(size) {
            val value = data.and(paridadeFinal)
            if (value!=0){
                count += 1
                HAL.setBits(SDX_VAL)
                HAL.setBits(SCLK_MASK)
                HAL.clrBits(SCLK_MASK)
            }
            else {
                HAL.clrBits(SDX_VAL)
                HAL.setBits(SCLK_MASK)
                HAL.clrBits(SCLK_MASK)
            }
            paridadeFinal *= 2
        }
        if (count % 2!=0)
            parity = SDX_VAL
        HAL.writeBits(SDX_VAL, parity)
        HAL.setBits(SCLK_MASK)
        HAL.clrBits(SCLK_MASK)
        HAL.setBits(adress)
    }
}
```

G. Código Kotlin *ScoreDisplay*

```
import isel.leic.utils.Time

object ScoreDisplay { // Controla o mostrador de pontuação.
    const val SS_SCORE = 0x02
    const val SCLK_SCORE = 0x10
    const val SDX_SCORE = 0x08

    // Inicia a classe, estabelecendo os valores iniciais.
    fun init() {
        SerialEmitter.init()
    }

    // Envia comando para atualizar o valor do mostrador de pontuação
    fun setScore(value: Int) { //ainda n esta validado
        val tamanho = value.toString().length
        var number = tamanho - 1
        var count = 0
        while (count < tamanho) {
            val bin = value.toString()[count].digitToInt()
            val valor = bin.shl(3) + number
            SerialEmitter.send(SerialEmitter.Destination.SCORE, valor,7)
            number-=1
            count+=1
        }
        SerialEmitter.send(SerialEmitter.Destination.SCORE,0b0000110,7)
    }
    // Envia comando para desativar/ativar a visualização do mostrador de pontuação
    fun off(value: Boolean) {
        if (value) {
            SerialEmitter.send(addr = SerialEmitter.Destination.SCORE, 0b0001111 ,
7)
            //tem haver com a tabela do projeto Tabela1 - Modulo Score Display
        } else {
            SerialEmitter.send(addr = SerialEmitter.Destination.SCORE, 0b0000111 ,
7)
        }
    }
    fun restartScore(){
        var number = 5
        val bin = 0
        repeat(6){
            val valor = bin.shl(3) + number
            SerialEmitter.send(SerialEmitter.Destination.SCORE, valor,7)
            number-=1
        }
        SerialEmitter.send(SerialEmitter.Destination.SCORE,0b0000110,7)
    }
}
```

H. Código Kotlin *TUI*

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

object TUI {

    var collumL0 = 16
    var collumL1 = 16

    fun escrever(str: String, left: Boolean, lines: Int, coll: Int) {
        if (left) {
            LCD.cursor(lines, coll)
            LCD.write(str)
            if (lines == 0) {
                collumL0 += 1
            } else {
                collumL1 += 1
            }
        } else {
            LCD.cursor(lines, coll)
            LCD.write(str)
            if (lines == 0) {
                collumL0 -= 1
            } else {
                collumL1 -= 1
            }
        }
    }

    fun align(line : Int, col : Int, c : Char){
        align(line,col,c.toString())
    }
    fun align(line : Int, col : Int, txt : String){
        LCD.cursor(line,col)
        LCD.write(txt)
    }

    fun alignLeft(line : Int, c : Char) {
        alignLeft(line,c.toString())
    }

    fun alignLeft(line : Int, txt : String) {
      if(txt.length <= 15) {
          LCD.cursor(line, 0)
          LCD.write(txt)
      }
    }     fun alignRigth(line : Int, str : String) {
            if(str.length < LCD.COLS){
                LCD.cursor(line, LCD.COLS - str.length)
                LCD.write(str)
            }
    }


    fun alignMiddle(line : Int, c : Char) {
        alignMiddle(line,c.toString())
    }

    fun alignMiddle(line : Int, txt : String) {
```

```
        val halfTxt = LCD.COLS - txt.length
        if (halfTxt >= 0) {
            LCD.cursor(line, halfTxt / 2)
            LCD.write(txt)
        }
    }

    fun forWaitKey(timeout: Long) : Char{
        return KBD.waitKey(timeout)
    }

    fun clrLCD(){
        LCD.clear()
    }

    fun cursor(line:Int,col:Int){
        LCD.cursor(line,col)
    }

    fun init(){
        KBD.init()
        LCD.init()
    }
}
```

I. Código Kotlin *M*

```kotlin
object M {

    private const val MANUT_MASK = 0x80

    fun isM():Boolean{
        return HAL.isBit(MANUT_MASK)
    }
}
```

J. Código Kotlin *CoinAcceptor*

```
object CoinAcceptor {

    private const val COIN_MASK = 0x40
    private const val COIN_ACCEPT_MASK = 0x40


    fun init() {
        HAL.clrBits(COIN_ACCEPT_MASK)
    }

    fun isCoin(): Boolean {
        return HAL.isBit(COIN_MASK)
    }

    fun newCoin() {

        HAL.setBits(COIN_ACCEPT_MASK)
        HAL.clrBits(COIN_ACCEPT_MASK)
    }
}
```

K. Código Kotlin *FileAccess*

```kotlin
import java.io.File
import java.io.FileReader
import java.io.PrintWriter

object FileAccess {

    fun addPl(lista: MutableList<Scores.Player>){
        val file = FileReader("SIG_scores.txt")
        var linhas = file.readLines()
        var listaFicheiro = emptyList<Scores.Player>()

        for (i in 0 until linhas.size) {
            val split = linhas[i].split(";")
            val player = Scores.Player(split[0].toInt(), split[1])
            lista.add(player)
        }
        if(lista.size > 20) {
            val dif = lista.size - 20
            listaFicheiro = lista.dropLast(dif)
        }
        listaFicheiro = listaFicheiro.sortedByDescending { it.score }
        val pw = PrintWriter("SIG_scores.txt")
        for(line in listaFicheiro){
            pw.println("${line.score};${line.name}")
        }
        pw.close()
    }

    fun filechange(): List<Scores.Player> {
        val linhas = File("SIG_scores.txt").readLines()
        var listareturn = emptyList<Scores.Player>()
        for (i in 0 until linhas.size) {
            val listadelinha = linhas[i].split(";")
            val player = Scores.Player(listadelinha[0].toInt(), listadelinha[1])
            listareturn = listareturn + player
        }
        return listareturn
    }

    fun Mfile(coins: Int, games: Int) {
        val file = FileReader("mFile.txt")
        val pw = PrintWriter("mFile.txt")
        pw.println(games)
        pw.println(coins)
        pw.close()
    }

    fun lerFile(): List<String> {
        return File("mFile.txt").readLines()
    }
}
```

L. Código Kotlin *Scores*

```kotlin
object Scores {
    data class Player( val score: Int, val name: String)
    val lista = mutableListOf<Player>()

    fun init(){
        val scores = FileAccess.filechange()
        lista.addAll(scores)
        lista.sortByDescending {it.score}
    }

    fun getPlayers(): MutableList<Player> {
        return lista
    }
    fun atualizarstats(coins: Int, games: Int) {
        FileAccess.Mfile(coins, games)
    }
    fun addPlayer(nome: String , score:Int) {
        lista.add(Player(score,nome))
        lista.sortByDescending { it.score }
        if (lista.size > 20) {
            lista.removeAt(lista.size - 1)
        }
    }
    fun printPlayer(){
        FileAccess.addPl(lista)
    }
}
```

M. Código Kotlin *Statistics*

```kotlin
object Statistics {

    private var games = 0
    private var coins = 0
    private var credits = 0

    fun init() {
        credits = 0
        val stats = FileAccess.lerFile()
        games = stats[0].toInt()
        coins = stats[1].toInt()
    }

    fun incCoins(){
        coins++
        credits += 2
    }

    fun getGames():Int{
        return games
    }

    fun getCoins():Int{
        return coins
    }

    fun getCredits():Int{
        return credits
    }

    fun clearCount(){
        games = 0
        coins = 0
    }

    fun newGame(){
        games++
        credits--
    }
}
```

N. Código Kotlin *App*

```
import KBD.NONE
import TUI.collumL0
import TUI.collumL1
import isel.leic.utils.Time
import kotlin.random.Random

var lista0 = ""
var lista1 = ""
var score = 0
var gameOver = false
var tecla = ' '
const val INVADER = 0
const val SPACESHIP = 1

fun main() {
    init()
    menuDesign()
    do {
        if (M.isM()) {
            manutDisplay()
            menuDesign()
        } else if (tecla == '*' && !gameOver && !M.isM() && Statistics.getCredits()
> 0) {
            inGame()
            Statistics.newGame()
        } else if (!gameOver) {
            menuDesign()
        }
    } while (true)
}

fun menuDesign() {
    var i = 0
    showCoins()
    var bool = false
    while (true) {
        tecla = TUI.forWaitKey(1000)
//        if (bool == true && !CoinAcceptor.isCoin()){
//            CoinAcceptor.newCoin()
//            Statistics.incCoins()
//            showCoins()
//            bool = false
//        }
        if (CoinAcceptor.isCoin()) {
            //bool = true
            CoinAcceptor.newCoin()
            Statistics.incCoins()
            showCoins()
        } else if (tecla == '*' && Statistics.getCredits() > 0 || M.isM()) {
            break
        } else if (tecla == ' ' && !M.isM()) {
            if (Statistics.getCredits() != 0) {
                TUI.align(1, 0, "                    ")
                showCoins()
            }
            TUI.align(1, 0, "                    ")
            val linhas = Scores.getPlayers()
```

```
            if (i < 9){
                TUI.align(1, 0, "0${i+1}-${linhas[i].name}")
                TUI.alignRigth(1, "${linhas[i].score}")
            }
            else{
                TUI.align(1, 0, "${i+1}-${linhas[i].name}")
                TUI.alignRigth(1, "${linhas[i].score}")
            }
            if (!CoinAcceptor.isCoin()) {
                Time.sleep(1250)
            }
            if (i == linhas.size - 1) {
                i = -1
            }
            i++
        }
    }
}

fun insertNewName() {
    var alf = 'A'
    var nome = StringBuilder("A                ")
    var coluna = 5
    TUI.align(0, 0, "                ")
    TUI.align(0, 0, "Name:")
    TUI.align(0, coluna, "$alf")
    while (true) {
        TUI.cursor(0, coluna)
        val tecla = TUI.forWaitKey(1000)
        if (tecla == '2') {
            alf += 1
            if (alf == '[') alf = 'A'
            nome[coluna - 5] = alf
            TUI.align(0, coluna, "$alf")
        } else if (tecla == '8') {
            alf -= 1
            if (alf == '@') alf = 'Z'
            nome[coluna - 5] = alf
            TUI.align(0, coluna, "$alf")
        } else if (tecla == '4') {
            if (coluna - 5 >= 1) {
                coluna -= 1
                alf = nome[coluna - 5]
                TUI.align(0, coluna, "$alf")
            }
        } else if (tecla == '6') {
            if (coluna - 5 <= 9) {
                coluna += 1
                if (nome[coluna - 5] == ' ') {
                    nome[coluna - 5] = 'A'
                }
                alf = nome[coluna - 5]
                TUI.align(0, coluna, "$alf")
            }
        } else if (tecla == '5') {
            break
        } else if (tecla == '*' && coluna - 5 == nome.trimEnd().length - 1 && coluna
>= 6) {
            nome.deleteCharAt(coluna - 5)
            TUI.align(0, coluna, " ")
```

```
            coluna--
        }
    }
    val nomeSemEspacos = nome.toString().trim()
    val nomeFinal = if (nomeSemEspacos.length > 1) {
        nomeSemEspacos[0] + nomeSemEspacos.substring(1).lowercase()
    } else {
        nomeSemEspacos
    }
    Scores.addPlayer(nomeFinal, score)
}

fun manutDisplay(): Char {
    var key = ' '
    while (M.isM()) {
        TUI.clrLCD()
        TUI.alignMiddle(0, "On Maintenance")
        TUI.alignLeft(1, "*-Count #-shutD")
        key = TUI.forWaitKey(1000)
        if (key == '*') {
            TUI.clrLCD()
            TUI.alignLeft(0, "Games:${Statistics.getGames()}")
            TUI.alignLeft(1, "Coins:${Statistics.getCoins()}")
            key = TUI.forWaitKey(5000)
            if (key == '#') {
                TUI.clrLCD()
                TUI.alignMiddle(0, "Clear counters")
                TUI.alignLeft(1, "5-Yes  other-No")
                key = TUI.forWaitKey(5000)
                if (key == '5') {
                    Statistics.clearCount()
                }
            }
        } else if (key == '#') {
            TUI.clrLCD()
            TUI.alignMiddle(0, "Shutdown")
            TUI.alignLeft(1, "5-Yes  other-No")
            key = TUI.forWaitKey(5000)
            if (key == '5') {
                Scores.atualizarstats(Statistics.getCoins(), Statistics.getGames())
                Scores.printPlayer()
                System.exit(-1)
            }
        } else if (key != ' ') {
            inGame()
        }
    }
    return key
}

fun barras() {
    TUI.alignLeft(0, "]")
    TUI.alignLeft(1, "]")
}

fun gameDesign() {
    TUI.clrLCD()
    barras()
    TUI.align(0, 1, SPACESHIP.toChar())
}
```

```
fun generateNumbers() {
    val number = randomNumbers()
    val addToList = Random.nextBoolean()
    if (addToList) {
        lista0 += number
        TUI.escrever(lista0, false, 0, collumL0)
    } else {
        lista1 += number
        TUI.escrever(lista1, false, 1, collumL1)
    }
}

fun gameOver() {
    TUI.clrLCD()
    TUI.align(0, 0, "*** GAME OVER **")
    TUI.align(1, 0, "Score:$score")
    ScoreDisplay.off(false)
    ScoreDisplay.setScore(score)
    gameOver = false
    Time.sleep(2000)
    if (score > 0) {
        insertNewName()
    }
    ScoreDisplay.restartScore()
    tecla = ' '
}

fun changeLine(line: Boolean) {
    if (line) {
        TUI.align(0, 1, " ")
        TUI.align(1, 1, SPACESHIP.toChar())
        barras()
    } else {
        TUI.align(1, 1, " ")
        TUI.align(0, 1, SPACESHIP.toChar())
        barras()
    }
}

fun inGame() {
    lista0 = ""
    lista1 = ""
    collumL0 = 16
    collumL1 = 16
    score = 0
    var key = ' '
    var keyFire = NONE
    var timeout = 600L
    var line0 = true
    gameDesign()
    do {
        timeout += Time.getTimeInMillis()
        while (Time.getTimeInMillis() < timeout) {
            keyFire = TUI.forWaitKey(1000)
            if (keyFire != '#' && keyFire != NONE && keyFire != '*') {
                key = keyFire
                TUI.alignLeft(if (line0) 0 else 1, key)
            }
            if (keyFire == '*') {
```

```
                changeLine(line0)
                line0 = !line0
            }
            if (keyFire == '#') {
                TUI.alignLeft(if (line0) 0 else 1, ']')
                if (hitNumber(key, line0)) {
                    deleteNumber(line0)
                    updateScore(key)
                }
            }
        }
        generateNumbers()
        timeout = 600
    } while (lista0.length < 16 && lista1.length < 16)
    gameOver()
}

fun hitNumber(key: Char, line0: Boolean): Boolean {
    return if (line0)
        key == lista0[0]
    else key == lista1[0]
}


fun deleteNumber(line0: Boolean) {
    if (line0) {
        TUI.align(0, collumL0 + 1, " ")
        lista0 = lista0.drop(1)
        collumL0 += 1
        barras()
    } else {
        TUI.align(1, collumL1 + 1, " ")
        lista1 = lista1.drop(1)
        collumL1 += 1
        barras()
    }
}

fun updateScore(key: Char) {
    score += key.digitToInt() + 1
    ScoreDisplay.off(false)
    ScoreDisplay.setScore(score)
}

fun showCoins() {
    TUI.clrLCD()
    TUI.alignMiddle(0, " SPACE INVADERS")
    TUI.alignLeft(1, " GAME")
    TUI.align(1, 6, SPACESHIP.toChar())
    TUI.align(1, 9, INVADER.toChar())
    TUI.align(1, 11, INVADER.toChar())
    TUI.alignRigth(1, "$${Statistics.getCredits()}")
    Time.sleep(1250)
}

fun randomNumbers(): Int {
    return Random.nextInt(9)
}

fun init() {
```

```
    TUI.init()
    ScoreDisplay.init()
    Statistics.init()
    CoinAcceptor.init()
    Scores.init()
}
```