ISEL
Departamento de Engenharia Eletrónica e Telecomunicações e de Computadores

Licenciatura em Engenharia Informática e de Computadores

Jogo Invasores Espaciais (*Space Invaders Game*)

António Seara (A50632@alunos.isel.pt)

Projeto
de
Laboratório de Informática e Computadores
2023 / 2024
verão

31 de maio 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.

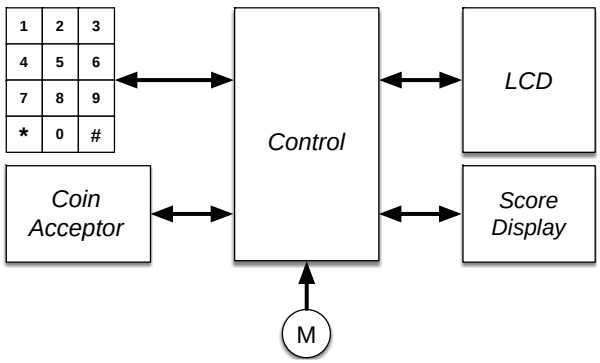

Figura 1 – Diagrama de blocos do jogo Invasores Espaciais (*Space Invaders Game*)

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*.

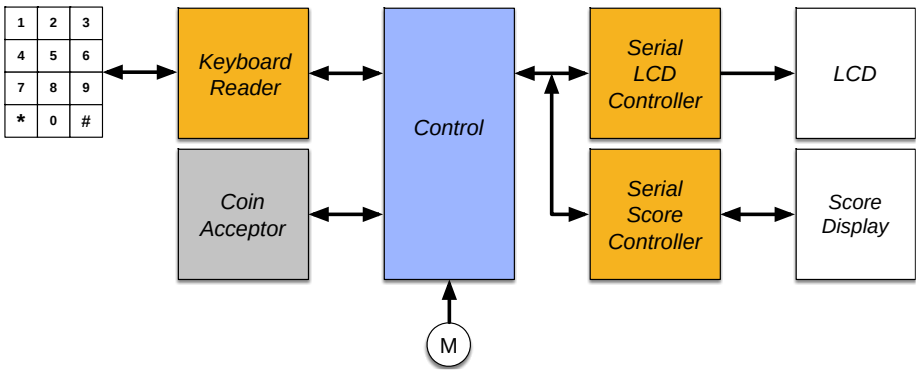

Figura 2 – Arquitetura do sistema que implementa o jogo Invasores Espaciais (*Space Invaders Game*)

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 Figura 3.

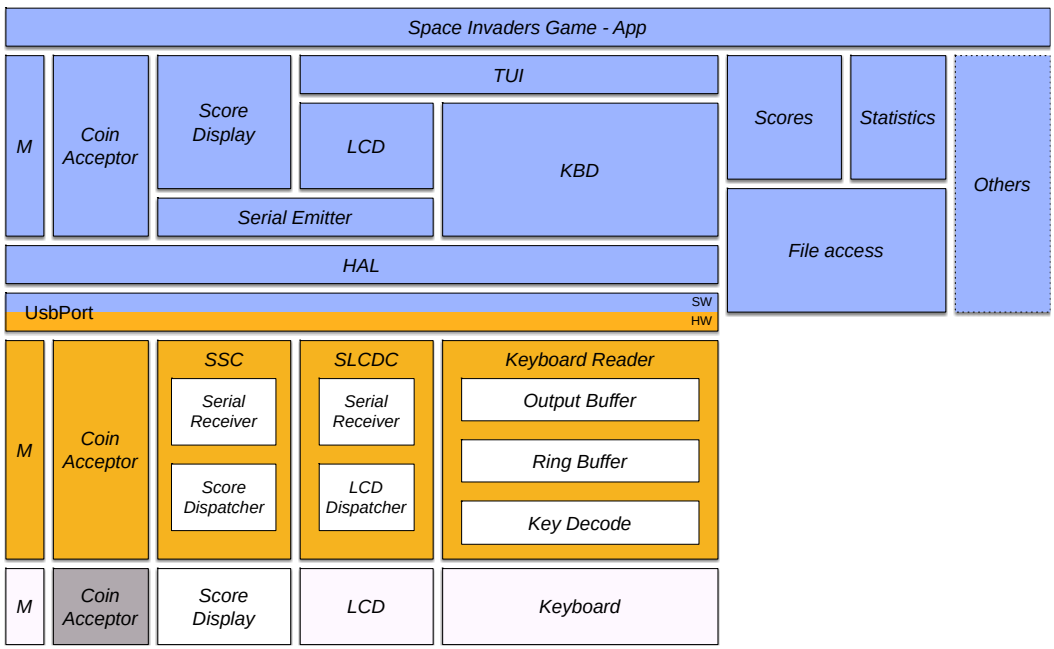

Figura 3 – Diagrama lógico do Jogo Invasores Espaciais (*Space Invaders Game*)

3 Implementação do sistema

A implementação do sistema é realizada com uma combinação de componentes de hardware e software. O hardware inclui um teclado de 12 teclas, um mostrador LCD de duas linhas e 16 caracteres, um mostrador de pontuação, um moedeiro, e uma chave de manutenção. O software é responsável por controlar a lógica do jogo, que é executada num PC utilizando a linguagem Kotlin.

O sistema é dividido nos seguintes módulos de hardware:

- **Keyboard Reader:** Descodifica o teclado e envia o código da tecla pressionada para o módulo de controle.
- **Serial LCD Controller (SLCDC):** Interface para comunicação com o mostrador LCD.
- **Serial Score Controller (SSC):** Interface para comunicação com o mostrador de pontuação.
- **Coin Acceptor:** Simula a aceitação de moedas.
- **Maintenance:** Simula verificação do switch de manutenção.

Os módulos de software são os seguintes:

- **HAL (Hardware Abstraction Layer):** Proporciona uma camada de abstração para o acesso ao hardware, permitindo a leitura e escrita de bits nas portas USB.
- **KBD (Keyboard):** Lida com a leitura das teclas pressionadas, retornando o código correspondente à tecla pressionada.
- **LCD:** Gerencia a escrita de dados no mostrador LCD, incluindo comandos e dados a serem exibidos.
- **SerialEmitter:** Envia tramas de dados para os módulos receptores seriais, identificando o destino e o número de bits a serem enviados.
- **ScoreDisplay:** Controla o mostrador de pontuação, atualizando e gerenciando a exibição da pontuação.
- **TUI (Text User Interface):** Gerencia a interface de usuário em texto.
- **M (Maintenance):** Gerencia as funções de manutenção do sistema.
- **CoinAcceptor:** Simula a aceitação de moedas e envia sinais para adicionar créditos ao jogo.
- **FileAccess:** Lida com o acesso e armazenamento persistente dos dados do sistema, incluindo contadores e pontuações.
- **Scores:** Gerencia a lista de pontuações, mantendo as 20 melhores pontuações.
- **Statistics:** Coleta e processa estatísticas sobre o jogo.
- **App:** O módulo principal que integra todos os outros módulos e gerencia o ciclo de vida do jogo.

Cada um desses módulos interage através de um protocolo de comunicação série síncrona, garantindo que os dados sejam transmitidos de forma eficiente entre o hardware e o software.

4 Conclusões

O projeto do jogo Invasores Espaciais demonstrou a viabilidade de integrar diversos componentes de hardware e software para criar um sistema de jogo funcional e interativo.

Durante o desenvolvimento do projeto, foram superados diversos desafios técnicos, especialmente na integração entre os componentes de hardware e software. A arquitetura modular adotada permitiu que cada componente fosse desenvolvido e testado de forma independente, garantindo maior robustez e facilidade na deteção de erros. Os módulos de hardware, como o keyboard reader, o SLCDC e o SSC, foram eficazmente integrados com os módulos de software, resultando em um sistema coeso e bem-sucedido.

Os testes realizados confirmaram que todos os componentes funcionam conforme o esperado. O LCD e o Score Display proporcionaram uma interface clara e eficiente para o jogador, melhorando a experiência do usuário. As funcionalidades de manutenção, incluindo a capacidade de dar reset aos contadores e realizar jogos de teste, adicionaram uma camada extra de controle e gerenciamento ao sistema.

O projeto também evidenciou a importância de uma boa documentação e planejamento. A criação de diagramas detalhados e a definição clara das interações entre os módulos facilitaram o desenvolvimento e a integração dos diferentes componentes. As várias avaliações intercalares permitiram a constante melhoria e ajuste do sistema, com feedback contínuo sendo incorporado ao longo do processo de desenvolvimento.

Concluindo, o projeto não só atingiu os objetivos técnicos, mas também proporcionou um momento de aprendizagem, demonstrando a importância da integração entre hardware e software em sistemas complexos e destacando a eficácia das abordagens modulares no desenvolvimento de sistemas robustos e escaláveis.

A. Interligações entre o HW e SW

Serial emitter:

- Mascara Clear = 110
- Mascara Data = 1
- Mascara LCD = 10
- Mascara SCORE = 100
- Mascara clock = 1000

Maintenance:

- Mascara Maintenance = 1000 0000

KBD:

- Mascara KVAL = 1 0000
- Mascara DATA = 1111
- Mascara KACK = 1000 0000

Coininserted:

- Mascara coinmask = 0100 0000

B. Atribuição de Pinos

```
#============================================================
# Altera DE10-Lite board settings
#============================================================


set_global_assignment -name FAMILY "MAX 10 FPGA"
set_global_assignment -name DEVICE 10M50DAF484C6GES
set_global_assignment -name TOP_LEVEL_ENTITY "DE10_Lite"
set_global_assignment -name DEVICE_FILTER_PACKAGE FBGA
set_global_assignment -name SDC_FILE DE10_Lite.sdc
set_global_assignment -name INTERNAL_FLASH_UPDATE_MODE "SINGLE IMAGE WITH ERAM"

#============================================================
# CLOCK
#============================================================
#set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to CLOCK_50
#set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to CLOCK2_50
#set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to CLOCK_ADC_10
#set_location_assignment PIN_P11 -to CLOCK_50
set_location_assignment PIN_N14 -to MCLK
#set_location_assignment PIN_N5 -to CLOCK_ADC_10

#============================================================
# SW
#============================================================
set_location_assignment PIN_C10 -to Manut
set_location_assignment PIN_C11 -to CoinIn
#set_location_assignment PIN_D12 -to SW[2]
#set_location_assignment PIN_C12 -to SW[3]
#set_location_assignment PIN_A12 -to SW[4]
#set_location_assignment PIN_B12 -to SW[5]
#set_location_assignment PIN_A13 -to SW[6]
#set_location_assignment PIN_A14 -to SW[7]
#set_location_assignment PIN_B14 -to SW[8]
set_location_assignment PIN_F15 -to Reset


#============================================================
# HEX0
#============================================================
#set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX0[0]
#set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX0[1]
#set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX0[2]
#set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX0[3]
#set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX0[4]
#set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX0[5]
#set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX0[6]
#set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX0[7]
set_location_assignment PIN_C14 -to HexLED0[0]
set_location_assignment PIN_E15 -to HexLED0[1]
set_location_assignment PIN_C15 -to HexLED0[2]
set_location_assignment PIN_C16 -to HexLED0[3]
set_location_assignment PIN_E16 -to HexLED0[4]
set_location_assignment PIN_D17 -to HexLED0[5]
set_location_assignment PIN_C17 -to HexLED0[6]
set_location_assignment PIN_D15 -to HexLED0[7]

#============================================================
```

```
# HEX1
#============================================================
#set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX1[0]
#set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX1[1]
#set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX1[2]
#set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX1[3]
#set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX1[4]
#set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX1[5]
#set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX1[6]
#set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX1[7]
set_location_assignment PIN_C18 -to HEXLED1[0]
set_location_assignment PIN_D18 -to HEXLED1[1]
set_location_assignment PIN_E18 -to HEXLED1[2]
set_location_assignment PIN_B16 -to HEXLED1[3]
set_location_assignment PIN_A17 -to HEXLED1[4]
set_location_assignment PIN_A18 -to HEXLED1[5]
set_location_assignment PIN_B17 -to HEXLED1[6]
set_location_assignment PIN_A16 -to HEXLED1[7]

#============================================================
# HEX2
#============================================================
#set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX2[0]
#set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX2[1]
#set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX2[2]
#set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX2[3]
#set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX2[4]
#set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX2[5]
#set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX2[6]
#set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX2[7]
set_location_assignment PIN_B20 -to HEXLED2[0]
set_location_assignment PIN_A20 -to HEXLED2[1]
set_location_assignment PIN_B19 -to HEXLED2[2]
set_location_assignment PIN_A21 -to HEXLED2[3]
set_location_assignment PIN_B21 -to HEXLED2[4]
set_location_assignment PIN_C22 -to HEXLED2[5]
set_location_assignment PIN_B22 -to HEXLED2[6]
set_location_assignment PIN_A19 -to HEXLED2[7]

#============================================================
# HEX3
#============================================================
#set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX3[0]
#set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX3[1]
#set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX3[2]
#set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX3[3]
#set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX3[4]
#set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX3[5]
#set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX3[6]
#set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX3[7]
set_location_assignment PIN_F21 -to HEXLED3[0]
set_location_assignment PIN_E22 -to HEXLED3[1]
set_location_assignment PIN_E21 -to HEXLED3[2]
set_location_assignment PIN_C19 -to HEXLED3[3]
set_location_assignment PIN_C20 -to HEXLED3[4]
set_location_assignment PIN_D19 -to HEXLED3[5]
set_location_assignment PIN_E17 -to HEXLED3[6]
set_location_assignment PIN_D22 -to HEXLED3[7]
```

```
#============================================================
# HEX4
#============================================================
#set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX4[0]
#set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX4[1]
#set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX4[2]
#set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX4[3]
#set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX4[4]
#set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX4[5]
#set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX4[6]
#set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX4[7]
set_location_assignment PIN_F18 -to HEXLED4[0]
set_location_assignment PIN_E20 -to HEXLED4[1]
set_location_assignment PIN_E19 -to HEXLED4[2]
set_location_assignment PIN_J18 -to HEXLED4[3]
set_location_assignment PIN_H19 -to HEXLED4[4]
set_location_assignment PIN_F19 -to HEXLED4[5]
set_location_assignment PIN_F20 -to HEXLED4[6]
set_location_assignment PIN_F17 -to HEXLED4[7]

#============================================================
# HEX5
#============================================================
#set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX5[0]
#set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX5[1]
#set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX5[2]
#set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX5[3]
#set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX5[4]
#set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX5[5]
#set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX5[6]
#set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX5[7]
set_location_assignment PIN_J20 -to HEXLED5[0]
set_location_assignment PIN_K20 -to HEXLED5[1]
set_location_assignment PIN_L18 -to HEXLED5[2]
set_location_assignment PIN_N18 -to HEXLED5[3]
set_location_assignment PIN_M20 -to HEXLED5[4]
set_location_assignment PIN_N19 -to HEXLED5[5]
set_location_assignment PIN_N20 -to HEXLED5[6]
set_location_assignment PIN_L19 -to HEXLED5[7]


#============================================================================
# Extension Board
#============================================================================

set_location_assignment PIN_W8     -to DoutLCD[0]
set_location_assignment PIN_V5     -to WrLOut

set_location_assignment PIN_AA15   -to DoutLCD[1]
set_location_assignment PIN_W13    -to DoutLCD[2]
set_location_assignment PIN_AB13   -to DoutLCD[3]
set_location_assignment PIN_Y11    -to DoutLCD[4]
set_location_assignment PIN_W11    -to DoutLCD[5]
set_location_assignment PIN_AA10   -to DoutLCD[6]
set_location_assignment PIN_Y8     -to DoutLCD[7]
set_location_assignment PIN_Y7     -to DoutLCD[8]

set_location_assignment PIN_W5     -to Linhas[0]
set_location_assignment PIN_AA14   -to      Linhas[1]
```

```
set_location_assignment PIN_W12    -to Linhas[2]
set_location_assignment PIN_AB12   -to      Linhas[3]
set_location_assignment PIN_AB11   -to      Col[0]
set_location_assignment PIN_AB10   -to      Col[1]
set_location_assignment PIN_AA9    -to      Col[2]
#set_location_assignment PIN_AA8    -to     KEYPAD_COL[3]

#============================================================
# End of pin and io_standard assignments
#============================================================
```

C. Código Kotlin *HAL*

```
import isel.leic.UsbPort

object HAL { // Virtualiza o acesso ao sistema UsbPort
    // Inicia a classe
    var Lastoutput = 0

    fun init() {
        Lastoutput = 0
        UsbPort.write(Lastoutput)

    }

    // Retorna true se o bit tiver o valor lógico ‘1’
    fun isBit(mask: Int): Boolean = UsbPort.read().and(mask) != 0


    // Retorna os valores dos bits representados por mask presentes no UsbPort
    fun readBits(mask: Int): Int = UsbPort.read() and mask

    // Escreve nos bits representados por mask os valores dos bits correspondentes em value
    fun writeBits(mask: Int, value: Int){

        Lastoutput = (Lastoutput and mask.inv()) or (value and mask)

        UsbPort.write(Lastoutput)
    }

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

    fun clrBits(mask: Int){
        Lastoutput = Lastoutput and mask.inv()
        UsbPort.write(Lastoutput )
    }
}
```

D. Código Kotlin *KBD*

```
import isel.leic.utils.Time

object KBD { // Ler teclas. Métodos retornam ‘0’..’9’,’#’,’*’ ou NONE.
    const val NONE = 0
    const val KVAL_MASK = 16
    const val K_DATA_MASK = 15
    const val KACK_MASK = 128

    // Inicia a classe
    fun init() {
        HAL.clrBits(KACK_MASK)
    }

    //Retorna de imediato a tecla premida ou NONE se não há tecla premida.
    fun getKey(): Char {
        val Kval = HAL.isBit(KVAL_MASK)
        if (Kval) {
            val value = HAL.readBits(K_DATA_MASK)
            val button = when (value) {
                0 -> '1'
                1 -> '4'
                2 -> '7'
                3 -> '*'

                4 -> '2'
                5 -> '5'
                6 -> '8'
                7 -> '0'

                8 -> '3'
                9 -> '6'
                10 -> '9'
                11 -> '#'
                else -> NONE.toChar()
            }

            HAL.setBits(KACK_MASK) //Kack está a TRUE

            while (HAL.isBit(KVAL_MASK));


            HAL.clrBits(KACK_MASK) // Kack está a false

            return button

        }
        return NONE.toChar()
    }

    // Retorna a tecla premida, caso ocorra antes do ‘timeout’ (representado em milissegundos), ou
NONE caso contrário.
    fun waitKey(timeout: Long): Char {
        val time = Time.getTimeInMillis() + timeout
        while(Time.getTimeInMillis() <= time) {
```

```
            val key = getKey()
            if (key != NONE.toChar())
                return key
        }
        return NONE.toChar()
    }
}

fun main(){
    HAL.init()
    KBD.init()
    while (true){
        val k = KBD.waitKey(10)
        if (k != KBD.NONE.toChar()){
            println(k)
        }
    }
}
```

E. Código Kotlin *LCD*

```
import isel.leic.utils.Time

object LCD {
    private const val LINES = 2
    private const val COLS = 16
    private const val rs_bit = 16
    private const val E_bit = 32
    private const val clk_reg_bit = 64

    fun init() {
        Time.sleep(150)

        writeCMD(48)
        Time.sleep(50)

        writeCMD(48)
        Time.sleep(10)

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

    }


    private fun writeByteParallel(rs: Boolean, data: Int){
        if(rs){
            HAL.setBits(rs_bit)
        }
        else{
            HAL.clrBits(rs_bit)
        }

        val shift_right = data.shr(4)

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



        HAL.writeBits(15, data)
        HAL.setBits(clk_reg_bit)
        HAL.clrBits(clk_reg_bit)

        HAL.setBits(E_bit)
        HAL.clrBits(E_bit)

    }

    private fun writeByteSerial(rs: Boolean, data: Int) {
```

```
        val temp_data:Int
        if(rs){
            temp_data = data.shl(1)+1
        }
        else{
            temp_data = data.shl(1)
        }
        SerialEmitter.send(SerialEmitter.Destination.LCD, temp_data, 10)
        Time.sleep(1)
    }

    private fun writeByte(rs: Boolean, data: Int){
        writeByteSerial(rs, data)
    }

    private fun writeCMD(data: Int){
        writeByte(false, data)
    }

    private fun writeDATA(data: Int){
        writeByte(true, data)
    }

    fun write(c: Char){
        writeDATA(c.code)
    }

    fun write(text: String){
        for (char in text){
            write(char)
        }
    }


    fun cursor(line: Int, column: Int){
        val write_data = 128
       writeCMD((line*0x40)+column+write_data)
    }

    fun clear(){
        writeCMD(1)
    }

    val spaceship = byteArrayOf(0b11110, 0b11000, 0b11100, 0b11111, 0b11100, 0b11000, 0b11110,
0b00000)

    val invader = byteArrayOf(0b11111, 0b11111, 0b10101, 0b11111, 0b11111, 0b10001, 0b10001,
0b00000)


    fun createCustomChar(location: Int, charmap: ByteArray) {
        // Set CGRAM address
        writeCMD(0x40 or (location shl 3))
        for (i in charmap.indices) {
            writeDATA(charmap[i].toInt())
        }
    }
```

```
    // Display the custom character (spaceship) on the LCD
    fun Ship(line: Int, column: Int) {
        // Create the custom character at location 0
        createCustomChar(0, spaceship)
        cursor(line, column)
        writeDATA(0) // Display the custom character stored at location 0
    }
    fun Invader(line:Int,column: Int){

        createCustomChar(2, invader)

        cursor(line, column)
        writeDATA(2) // Display the custom character stored at location 0
    }

}
fun main(){
    HAL.init()
    SerialEmitter.init()
    LCD.init()
    LCD.write("hello world")
}
```

F. Código Kotlin *SerialEmitter*

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

    // Inicia a classe
    fun init(){
        val mask_clr = 6
        HAL.setBits(mask_clr)
    }


// 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.
    fun send(addr: Destination, data: Int, size : Int){
        val mask_data = 1
        var counter = 0
        val mask_LCD = 2
        val mask_SCORE = 4
        val mask_clock = 8
        var data_temp = data

        if(addr == Destination.LCD)
            HAL.clrBits(mask_LCD)
        else if(addr == Destination.SCORE)
            HAL.clrBits(mask_SCORE)

        for(i in 0 until size-1){
            HAL.clrBits(mask_clock)
            if (data_temp%2 ==0){
                HAL.writeBits(mask_data, 0)
                data_temp = data_temp.shr(1)
            }
            else {
                HAL.writeBits(mask_data, 1)
                data_temp = data_temp.shr(1)
                counter++
            }
            HAL.setBits(mask_clock)
        }

        HAL.clrBits(mask_clock)
        if (counter%2==0){
            HAL.writeBits(mask_data, 0)
        }else{
            HAL.writeBits(mask_data, 1)
        }
        HAL.setBits(mask_clock)
        HAL.setBits(mask_LCD)
        HAL.setBits(mask_SCORE)
    }
}
```

G. Código Kotlin *ScoreDisplay*

```
object ScoreDisplay { // Controla o mostrador de pontuação.
    val updatemask = 96
    // Inicia a classe, estabelecendo os valores iniciais.
    fun init() {
        off(false)
        for (command in 5 downTo 0)
            sendScoreBit(0b1111 + command.shl(4))
            sendScoreBit(updatemask)

    }
    private fun sendScoreBit(value:Int){
        SerialEmitter.send(SerialEmitter.Destination.SCORE, value, 8)
    }


    // Envia comando para atualizar o valor do mostrador de pontuação
    fun setScore(value: Int){
        var temp_value = value
        var divider = 100_000
        var encounteredNonZero = false

        for(command in 5 downTo 0){
            val sub = (temp_value/divider)*divider
            if (temp_value/divider != 0 || encounteredNonZero){
                encounteredNonZero = true
                sendScoreBit((temp_value/divider)+command.shl(4))}
            else
                sendScoreBit((0b1111)+command.shl(4))
            temp_value -= sub
            divider /= 10
        }
        sendScoreBit(updatemask)
    }

    // Envia comando para desativar/ativar a visualização do mostrador de pontuação
    fun off(value: Boolean){
        val maskon = 112
        val maskoff = 113
        if(value){
            sendScoreBit(maskoff)}
        else{
            sendScoreBit(maskon)
        }
    }
}
fun main(){
    SerialEmitter.init()
    ScoreDisplay.init()
    ScoreDisplay.setScore(123)
}
```

H. Código Kotlin *TUI*

```
vvvobject TUI {

    fun write(top: String, bottom: String){
        LCD.clear()
        LCD.cursor(0,0)
        LCD.write(top)
        LCD.cursor(1,0)
        var pos = 0
        for (i in bottom) {
            if (i == '$'){
                Invader(1, pos)
            }else{
            LCD.write(i)}
            pos += 1
        }
    }
    fun write_question(question:String, option1:String, option2: String){
        LCD.clear()
        LCD.cursor(0,0)
        LCD.write(question)
        LCD.cursor(1,0)
        LCD.write("$option1 $option2")
    }
    fun pressed_key(): Char {
        return KBD.waitKey(10)
    }

    fun write_main(coins:Int){
        LCD.clear()
        val cointext = "$$coins"
        val text = " Game $  $ $    ".dropLast(cointext.length)
        write(" Space Invaders", text)
        LCD.write(cointext)
    }

    //primeiro elemento da direita para a esquerda que é uma empty string
    fun lastEmptyStringIndex(list: MutableList<String>): Int {
        for (i in list.size - 1 downTo 0) {
            if (list[i].isEmpty()) {
                return i
            }
        }
        return -1
    }
    //primeiro elemento da esquerda para a direita que é uma empty string
    fun firstEmptyStringIndex(list: MutableList<String>): Int {
        for (i in list.indices) {
            if (list[i].isEmpty()) {
                return i
            }
        }
        return 0
    }
    //primeiro elemento da direita para a esquerda antes de uma empty string
    fun firstElementBeforeEmptyFromRight(list: MutableList<String>): Int {
        for (i in list.size - 1 downTo 1) {
            if (list[i - 1].isEmpty() && list[i].isNotEmpty()) {
                return i
```

```
        }
    }
    return list.size-1
}
//troca um valor empty string com o elemento seguinte
fun switch(idx:Int, list:MutableList<String>){
    list[idx] = list[idx+1]
    list[idx+1] = ""
}
//update display
fun update(){
    var inc = 0
    LCD.clear()
    for (i in LCD0){
        LCD.cursor(0,inc)
        if (i == ">")
            Ship(0, inc)
        else
            LCD.write(i)
        inc++
    }
    inc =0
    for (i in LCD1){
        LCD.cursor(1,inc)
        if (i == ">")
            Ship(1, inc)
        else
            LCD.write(i)
        inc++
    }
    update=false
}
//update player
fun updatePlayer(aim:Char){
    LCD.clear()
    if (linha == 0){
        LCD0[0]=""
        LCD0[1]=""
        writeright(aim)
        writeright('>')
    }
    else if (linha == 1){
        LCD1[0]=""
        LCD1[1]=""
        writeright(aim)
        writeright('>')
    }
}

fun clear(){
    LCD.clear()
}

fun Ship(line: Int, column: Int) {
    LCD.Ship(line, column)
}
fun Invader(line:Int,column: Int){
    LCD.Invader(line, column)
}
```

```
}
```

I. Código Kotlin *M*

```
object M {
    fun ismaintenancebit(): Boolean {
        return HAL.isBit(128)
    }
}
```

J. Código Kotlin *CoinAcceptor*

```
object CoinAcceptor {


    var prevState = 0 // Initialize previous state to 0
    fun coininserted(): Boolean {
        val coinmask = 0b01000000
        val now = HAL.readBits(coinmask)

        if (prevState == 0 && now == coinmask) {
            coins += 1
            creds +=2
            HAL.setBits(coinmask)
            HAL.clrBits(coinmask)
            prevState = now // Update the previous state
            return true
        }

        prevState = now // Update the previous state
        return false
    }
}
```

K. Código Kotlin *FileAccess*

L. Código Kotlin *Scores*

```
import java.io.BufferedReader
import java.io.File
import java.io.FileReader

object Scores {

    fun read(filePath:String = "SIG_scores.txt"): MutableMap<Int, List<String>> {
        val br = BufferedReader(FileReader(filePath))
        var line = br.readLine()
        val highscores = mutableMapOf<Int, List<String>>()
        var key = 1
        while (line != null) {
            val formap = line.split(";","\n")
            highscores[key] = formap
            key ++
            line = br.readLine()
        }
        br.close()
        return highscores

    }

    fun generateScoreNameString(data: Map<Int, List<Any>>, score: Int, name: String): String {
        // Initialize a mutable list to collect the entries as pairs of (score, name)
        val entries = mutableListOf<Pair<Int, String>>()

        // Add existing entries from the map to the list
        for ((_, value) in data) {
            // value is expected to be a list with [score, name]
            val entryScore = value[0].toString().toInt()
            val entryName = value[1].toString()

            entries.add(Pair(entryScore, entryName))
        }

        // Add the new score and name to the list
        entries.add(Pair(score, name))

        // Sort the list by score in descending order
        entries.sortByDescending { it.first }

        // Ensure the list does not exceed 20 entries by removing the lowest scores
        if (entries.size > 20) {
            entries.subList(20, entries.size).clear()
        }

        // Convert the list of pairs into the desired string format
        val resultStrings = entries.map { "${it.first};${it.second}" }

        // Join all strings with a newline character and return the result
        return resultStrings.joinToString("\n")
    }


    fun write(score:Int,name:String, filePath:String = "SIG_scores.txt"){
        val highscores = read()
        val output = generateScoreNameString(highscores, score, name)
        val file = File(filePath)
```

```
        file.writeText(output)
    }
}
```

M. Código Kotlin *Statistics*

```
import java.io.BufferedReader
import java.io.File
import java.io.FileReader
import kotlin.math.ceil

object statistics {
    fun read(filePath:String = "statistics.txt"): Set<Int> {
        val br = BufferedReader(FileReader(filePath))
        var line = br.readLine()
        val numbers = mutableListOf<String>()
        while (line != null) {
            numbers += line.split("\n")
            line = br.readLine()
        }
        br.close()
        if (numbers.size >= 2) {
            return setOf(numbers[0].toInt(),numbers[1].toInt())
        } else {
            return setOf(0,0)
        }

    }

    fun write(games:Int, coins:Int, filePath:String = "statistics.txt"){
        val file = File(filePath)
        file.writeText("$games\n${coins}")
    }

}
```

N. Código Kotlin *App*

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



//cria enimigos a direita e mexe os para a esquerda, verifica colisao com player
fun spawnEnemy(){
    val enemy = listOf('0', '1', '2', '3', '4', '5', '6', '7', '8', '9').random()

    val enemyline = (0..1).random()
    val col = when (enemyline) {
        0 -> TUI.lastEmptyStringIndex(LCD0)
        1 -> TUI.lastEmptyStringIndex(LCD1)
        else -> -1 }
    if (col ==-1){
        gameover =true
        return
    }
    if (enemyline ==0){
        for (idx in col..<LCD0.size-1)
            TUI.switch(idx, LCD0)
        LCD0[LCD0.size-1] = enemy.toString()
    }
    else if (enemyline ==1){
        for (idx in col..<LCD1.size-1)
            TUI.switch(idx, LCD1)
        LCD1[LCD1.size-1] = enemy.toString()
    }
}

//escreve a direita no primeiro espaço disponivel
fun writeright(value : Char){
    if(value != KBD.NONE.toChar() && value != '*' && value != '#'){
        val col = when (linha) {
            0 -> TUI.firstEmptyStringIndex(LCD0)
            1 -> TUI.firstEmptyStringIndex(LCD1)
            else -> -1 }
```

```
        if (linha ==0){
            LCD0[col] = value.toString()
        }
        else if (linha ==1){
            LCD1[col] = value.toString()
        }
    }
}

fun isHighscore(score: Int, scoreMap: MutableMap<Int, List<String>>): Boolean {
    // Check if the map size is smaller than 20
    if (scoreMap.size < 20) {
        return true
    }

    // Check if the score is higher than any of the scores in the map
    for ((_, value) in scoreMap) {
        val storedScore = value[0].toIntOrNull()
        if (storedScore != null && score > storedScore) {
            return true
        }
    }

    // If neither condition is met, return false
    return false
}

//menu que mostra score
fun scoremenu(score:Int){
    TUI.write("Score:$score", "")
    val highscores = Scores.read()
    val timeinit = Time.getTimeInMillis()
    var time:Long = Time.getTimeInMillis()
    var blinkscore = true

    val updateInterval = 250L // Update interval in milliseconds
    var lastUpdateTime = timeinit
```

```
    while (time - timeinit <= 3000) {
        time = System.currentTimeMillis()

        if (time - lastUpdateTime >= updateInterval) {
            if (blinkscore) {
                ScoreDisplay.off(true)
            } else {
                ScoreDisplay.off(false)
                ScoreDisplay.setScore(score)
            }
            blinkscore = !blinkscore
            lastUpdateTime = time
        }
    }
    if (isHighscore(score, highscores)){
        TUI.clear()
        highscoremenu(score)
    }
}

//menu que permite escrever um nome de 4 letras
fun highscoremenu(score: Int){
    LCD.cursor(0,0)
    val alfabeto = ('A'..'Z').toList()
    var idx = 0
    var col = 5
    val name = "        ".toMutableList()
    LCD.write("Name:A")
    while(true){
        val c = KBD.waitKey(1)
        if (c == '5') break
        if (c=='6'){
            col++
            if (col>= 13) col =5
            LCD.cursor(0, col)
            LCD.write(alfabeto[idx])
            name[col-5] = alfabeto[idx]
        }
```

```
        if (c=='4'){
            col--
            if (col<= 4) col =12
            LCD.cursor(0, col)
            LCD.write(alfabeto[idx])
            name[col-5] = alfabeto[idx]
        }
        if (c == '2'){
            idx = if (idx == alfabeto.size - 1) 0 else idx+1
            LCD.write(alfabeto[idx])
            LCD.cursor(0, col)
            name[col-5] = alfabeto[idx]

        }
        if (c == '8'){
            idx = if (idx == 0) alfabeto.size - 1 else idx-1
            LCD.write(alfabeto[idx])
            LCD.cursor(0, col)
            name[col-5] = alfabeto[idx]

        }
    }
    Scores.write(score, name.joinToString(""))
}

//retirar enimigo atingido do ecra
fun kill(list:MutableList<String>, aim:Char): String {
    val idx = TUI.firstElementBeforeEmptyFromRight(list)
    val enemy = list[idx]
    if(aim.toString() == enemy) list[idx] = ""
    return enemy
}

//mudar linha
fun changeLine(){
    if (linha==1){
        linha = 0
        writeright(']')
```

```
        writeright('>')
        LCD1[0]=""
        LCD1[1]=""
        update=true}
    else if (linha==0){
        linha = 1
        writeright(']')
        writeright('>')
        LCD0[0]=""
        LCD0[1]=""
        update=true}
}



fun formathighscore(number: Int, highscores: MutableMap<Int, List<String>>): String {
    return    "${number}-${highscores[number]?.get(1)}"+"                                    ".drop(number.toString().length+
(highscores[number]?.get(1)?.length
        ?: 0) + (highscores[number]?.get(0).toString().length)) + (highscores[number]?.get(0) ?: 0)
}



//iniciar o jogo
fun initMenu(){
    val highscores = Scores.read()

    val displayTime = 3000 // 5000 milliseconds = 5 seconds
    var startTime = System.currentTimeMillis()
    var showMessage1 = true
    val manutMask = 128
    var key = '~'
    TUI.write_main(creds)
    var number =1
    while (!(key == '*' && creds != 0)) {
        key = KBD.waitKey(10)
        if (CoinAcceptor.coininserted()){
            TUI.write_main(creds)
            showMessage1 = true
```

```
        }
        if (HAL.isBit(manutMask)){
            manut()
        }
        val currentTime = System.currentTimeMillis()

        if (currentTime - startTime >= displayTime) {
            if (showMessage1) {
                TUI.write(" Space Invaders", formathighscore(number, highscores))
                number ++
                if (number>highscores.size) number =1
            } else {
                TUI.write_main(creds)
            }
            // Reset the timer and switch the message
            startTime = currentTime
            showMessage1 = !showMessage1
        }
    }
    creds-=1
}
val statisticsSET = statistics.read()
var games = statisticsSET.first()
var coins = statisticsSET.last()
var creds = 0



fun manut(){

    TUI.write(" On Maintenance", "*-Count #-shutD")

    while (M.ismaintenancebit()) {
        var clear = false

        var key = TUI.pressed_key()
        if(key == '*'){
            val statisticsSET = statistics.read()
```

```
    games = statisticsSET.first()

    TUI.write("Games:$games", "Coins:$coins")
    val timeinit = Time.getTimeInMillis()
    var time:Long = 0

    while (time-timeinit<=5000){
        time = Time.getTimeInMillis()
        key = TUI.pressed_key()
        if(key == '#'){
            TUI.write_question(" Clear Counters", "5-Yes", "Other-No")
            clear = true
        }
        else if(key == '5' && clear){
            statistics.write(0, 0)
            break
        }
        else if(key in listOf('0', '1', '2', '3', '4', '6', '7', '8', '9', '*')){
            break
        }
    }
    TUI.write(" On Maintenance", "*-Count #-shutD")
}
else if(key == '#'){
    TUI.write_question("    Shutdown", "5-Yes", "other-No")
    val timeinit = Time.getTimeInMillis()
    var time:Long = 0

    while (time-timeinit<=5000){
        time = Time.getTimeInMillis()
        key = TUI.pressed_key()
        if (key == '5'){
            statistics.write(games, coins)
            exitProcess(0)
        }
        else if (key in listOf('0', '1', '2', '3', '4', '6', '7', '8', '9', '*', '#')){
            break
        }
```

```
            }
            TUI.write(" On Maintenance", "*-Count #-shutD")
        }
            else if(key in listOf('0', '1', '2', '3', '4', '5', '6', '7', '8', '9')){
                gameloop()
                TUI.write(" On Maintenance", "*-Count #-shutD")
            }
    }
    }

    var linha = 0
    var LCD0 = mutableListOf("", "", "", "", "", "", "", "", "", "", "", "", "", "", "", "")
    var LCD1 = mutableListOf("", "", "", "", "", "", "", "", "", "", "", "", "", "", "", "")
    var gameover = false
    //flag para saber se houve alteraçao para dar update ao LCD
    var update = false
    var score = 0


    fun gameloop(){

        var aim = ']'
        var timeinit = Time.getTimeInMillis()
        var key: Char
        if (!M.ismaintenancebit()) {
            initMenu()
        }
        writeright(']')
        writeright('>')

        TUI.update()

        while (!gameover) {
            key = KBD.waitKey(10)

            //guardar numero da aim
            if (key.digitToIntOrNull() != null){
```

```
        aim = key
        TUI.updatePlayer(aim)
        update = true
    }

    //update display
    if (update) TUI.update()

    //disparar
    if (key == '*' && aim != ']' ) {
        var enemy = "0"
        if (linha ==0){
            enemy = kill(LCD0, aim)
        }
        else if (linha ==1){
            enemy = kill(LCD1, aim)
        }
        aim = ']'
        TUI.updatePlayer(aim)
        score += enemy.toInt()+1
        ScoreDisplay.setScore(score)
        update=true
    }

    if (key == '#') changeLine()

    val time = Time.getTimeInMillis()
    //criar enimigo
    if (time-timeinit > 500.toLong()){
        spawnEnemy()
        timeinit=time
        update = true
    }

}
TUI.clear()
if (!M.ismaintenancebit()){
    scoremenu(score)
```

```
    }
}

fun main(){
    HAL.init()
    KBD.init()
    LCD.init()
    SerialEmitter.init()
    ScoreDisplay.init()
    while (true){
        gameloop()

        //reiniciar as variaveis
        games += 1
        TUI.clear()
        gameover = false
        score = 0
        ScoreDisplay.setScore(score)
        LCD0 = mutableListOf("", "", "", "", "", "", "", "", "", "", "", "", "", "", "", "")
        LCD1 = mutableListOf("", "", "", "", "", "", "", "", "", "", "", "", "", "", "", "")

    }
}
```