ISEL
ADEETC
Área Departamental de Engenharia Electrónica e de Telecomunicações e de Computadores

Licenciatura em Engenharia Informática e de Computadores

Jogo Invasores Espaciais (*Space Invaders Game*)

Avaliação intercalar I

João Nunes (A47220@alunos.isel.pt)
Alexandre Luís (A47222@alunos.isel.pt)
Miguel Marques (A47204@alunos.isel.pt)

Projeto
de
Laboratório de Informática e Computadores
2019 / 2020 verão

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 gerador de sons (*Sound Generator*) 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.

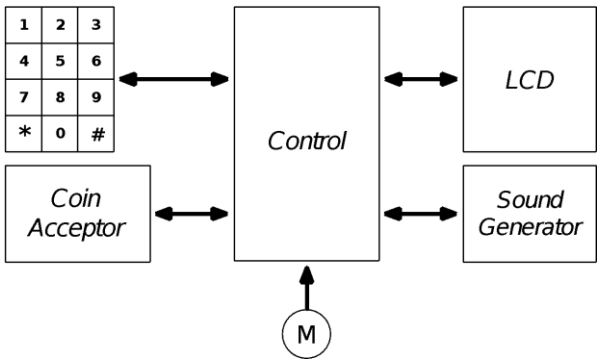


Figura 1 – Diagrama de blocos do jogo Invasores Espaciais (*Space Invaders Game*)

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*. 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 foi 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 gerador de sons (*Sound Generator*), designado por *Serial Sound Controller* (*SSC*); *iv*) um moedeiro, designado por *Coin Acceptor*; e *v*) um módulo de controlo, designado por *Control*. Os módulos *i*), *ii*) e *iii*) foram implementados em *hardware*, o moedeiro é simulado, enquanto o módulo de controlo foi implementado em *software* a executar num PC usando linguagem Java.

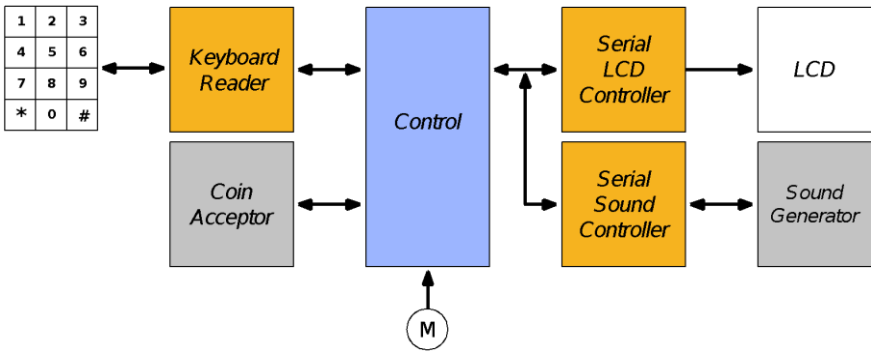


Figura 2 – Arquitetura do sistema que implementa o jogo Invasores Espaciais (*Space Invaders Game*)

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 dois códigos. O módulo *Control* processa os dados e envia a informação a apresentar no *LCD* através do módulo *SLCDC*. O gerador de sons é atuado pelo módulo *Control*, através do módulo *SSC*. Por razões de ordem física, e por forma a minimizar o número de fios de interligação, 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 Java e seguindo a arquitetura lógica apresentada na Figura 3.

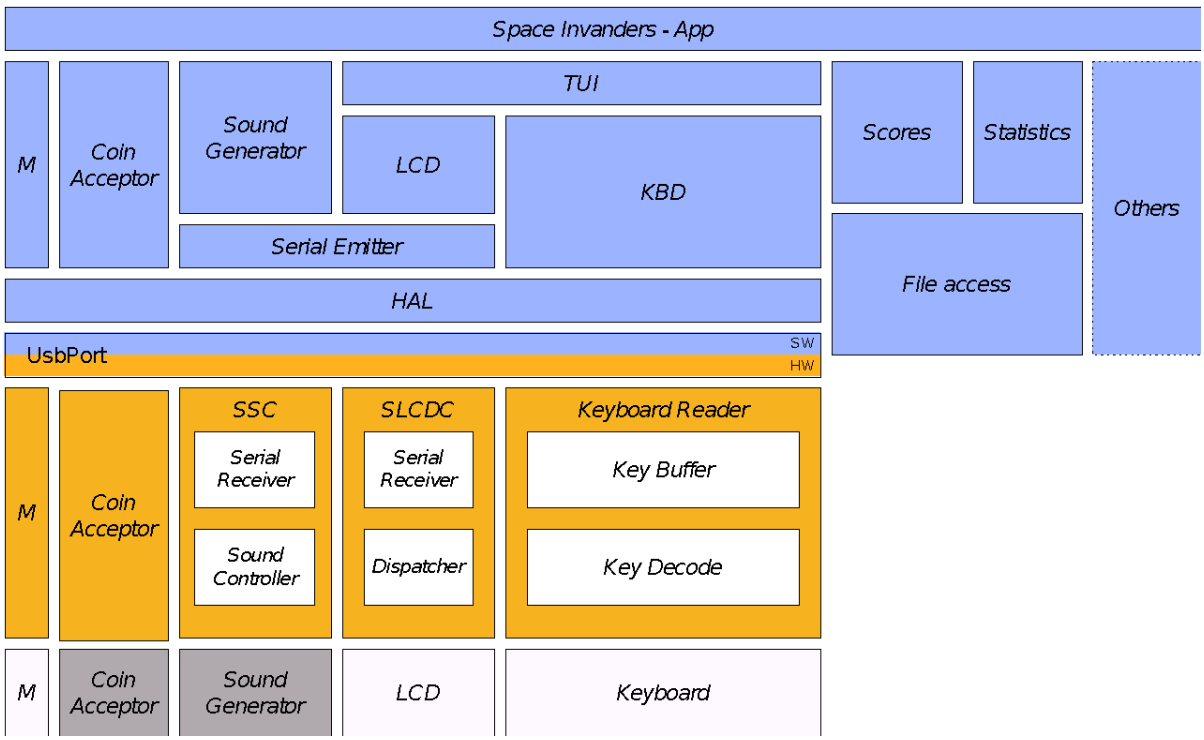
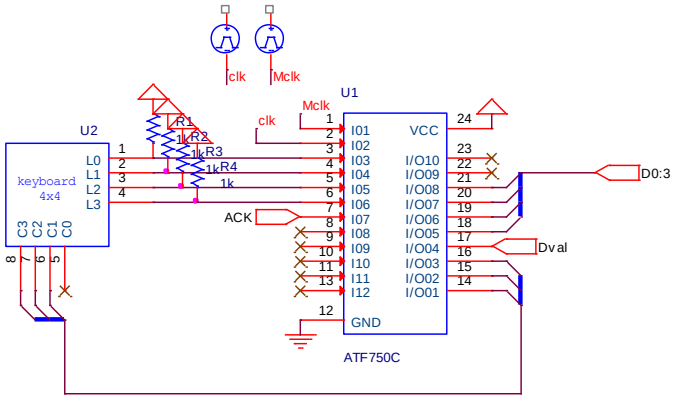


Figura 3 – Diagrama lógico do Jogo Invasores Espaciais (*Space Invaders Game*)

A. Interligações entre o HW e SW

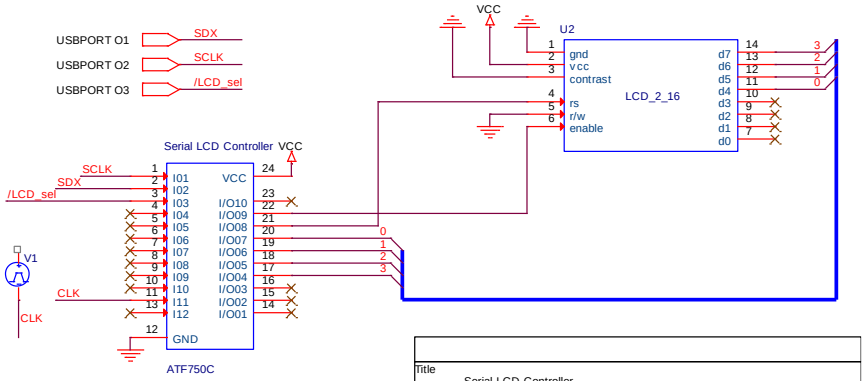
A realização das interligações do HW e SW feita em laboratório consta do seguinte esquema:

- *Keyboard Reader:*



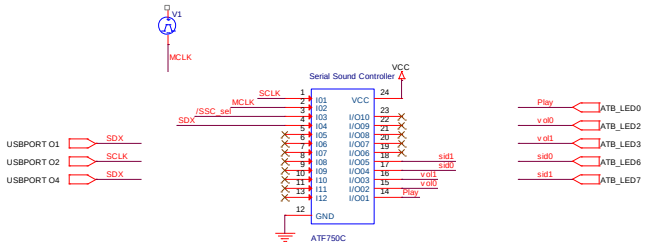
Title: KeyBoard Reader		
Size: A	Document Number: 1	Rev: 2
Date: Monday, June 01, 2020	Sheet 1 of 1	

- *Serial LCD Controller:*



Title: Serial LCD Controller		
Size: A	Document Number: 2	Rev: 1
Date: Wednesday, June 24, 2020	Sheet 1 of 1	

- *Serial Sound Controller:*



Title: Serial Sound Controller		
Size: A	Document Number: <Doc>	Rev: <RevCode>
Date: Thursday, July 02, 2020	Sheet 1 of 1	

B. Código Java da classe *HAL*

```
package edu.isel.lic.link;

import isel.leic.*;
import isel.leic.utils.Time;

import java.util.Scanner;

public class HAL // Virtualiza o acesso ao sistema UsbPort
{
  public static int input, output;
  public static final int MAX_BITS = 0xFF;
  private static final boolean ULICX = false; // mudar para true caso estiver a ser usado uLICx

  // Inicia a classe
  public static void init() {
    clrBits(MAX_BITS);
  }

  // Retorna true se o bit tiver o valor lógico '1'
  public static boolean isBit(int mask) {
    return (mask == readBits(mask));
  }

  // Retorna os valores dos bits representados por mask presentes no UsbPort
  public static int readBits(int mask) {
    getInput();
    return mask & input;
  }

  // Escreve nos bits representados por mask o valor de value
  public static void writeBits(int mask, int value) {
    output = mask & value | ~mask & output;
    updateOutput();
  }

  // Coloca os bits representados por mask no valor lógico '1'
  public static void setBits(int mask) {
    output = output | mask;
    updateOutput();
  }

  // Coloca os bits representados por mask no valor lógico '0'
  public static void clrBits(int mask) {
    output = output & ~mask;
    updateOutput();
  }

  // Atualiza a saída no UsbPort com o valor da variável output
  private static void updateOutput() {
    UsbPort.out(ULICX ? output : ~output);
  }

  // Atualiza a variável input com a entrada do UsbPort
  private static void getInput() {
    input = (ULICX) ? UsbPort.in() : ~UsbPort.in();
  }
}
```

C. Código Java da classe *KBD*

```
package edu.isel.lic.peripherals;

import edu.isel.lic.link.HAL;
import isel.leic.utils.Time;

public class KBD // Ler teclas. Métodos retornam '0'..'9','A'..'F' ou NONE.
{
  public static final char NONE = 0x20; // Para podermos usar as posições iniciais da CGRAM
  public static final String kbd="147*2580369#";
  public static final int Dval_MASK = 0x10, DATA_MASK = 0x0F, ACK_MASK = 0x20;

  // Inicia a classe
  public static void init() {
    HAL.clrBits(ACK_MASK);
  }

  // Retorna de imediato a tecla premida ou NONE se não há tecla premida.
  public static char getKey() {
    char key = NONE;

    if (HAL.isBit(Dval_MASK)) {
      key = kbd.charAt(HAL.readBits(DATA_MASK));
      HAL.setBits(ACK_MASK);
      //Time.sleep(1);
      HAL.clrBits(ACK_MASK);
    }

    return key;
  }

  // Retorna quando a tecla for premida ou NONE após decorrido 'timeout' milisegundos.
  public static char waitKey(int timeout) {
    char key;
    long starting_time = Time.getTimeInMillis(), current_time;
    boolean keyPress = false;

    do {
      current_time = Time.getTimeInMillis();
      key = getKey();
      if (key != NONE)
        keyPress = true;
    } while((current_time - starting_time < timeout) && !keyPress);

    return key;
  }
}
```

D. Código Java da classe *SerialEmitter*

```
package edu.isel.lic.link;

import edu.isel.lic.link.sound.SoundGenerator;
import edu.isel.lic.peripherals.lcd.LCD;
import isel.leic.utils.Time;

public class SerialEmitter { // Envia tramas para os diferentes módulos Serial Receiver.

   private static final int SLCDC_SELECT_MASK = 0x08, SSC_SELECT_MASK = 0x10, DATA_MASK = 0x02;
   public static final int CLOCK_MASK = 0x04;

   public enum Destination {
      SLCD(SLCDC_SELECT_MASK), SSC(SSC_SELECT_MASK);

      private final int addr;

      Destination (int addr) {
         this.addr = addr;
      }
      public int getAddr () {
         return addr;
      }
   }

   // Usado para ativar testes específicos
   private static final boolean lcd_test = false;
   private static final boolean sound_test = true;

   // Para efeitos de teste
   public static void main (String[] args) {
      HAL.init();
      init();

      if (lcd_test) {
         send(Destination.SLCD, LCD.SERIAL_DATA_SIZE, 0b10101);
//count of 1 bits = 3 | parity bit = 0
         Time.sleep(2000);
         send(Destination.SLCD, LCD.SERIAL_DATA_SIZE, 0b11000);
//count of 1 bits = 2 | parity bit = 1
      }
      if (sound_test) {
         send(Destination.SSC, SoundGenerator.SERIAL_DATA_SIZE, 0b1010);
         Time.sleep(2000);
         send(Destination.SSC, SoundGenerator.SERIAL_DATA_SIZE, 0b1111);
         Time.sleep(2000);
         send(Destination.SSC, SoundGenerator.SERIAL_DATA_SIZE, 0b0001);
      }
   }

   // Inicia a classe
   public static void init() {
HAL.setBits(Destination.SLCD.getAddr() | Destination.SSC.getAddr()); }

   // Envia uma trama para o SerialReceiver identificado por addr, com a dimensão de size e os bits de 'data'.
   public static void send(Destination addr, int size, int data) {
      int select_mask = addr.getAddr(), bit_counter = 0;

      HAL.clrBits(select_mask);
      for (int current_bit = 0; current_bit < size; current_bit++) {
         if ((data & (1 << current_bit)) != 0) {
            HAL.setBits(DATA_MASK);
            bit_counter++;
         }
         else
            HAL.clrBits(DATA_MASK);
         clockPulse();
      }

      HAL.writeBits(DATA_MASK,parityCheck(bit_counter));
      clockPulse();
      HAL.setBits(select_mask);
   }

   // Envia um bit a '1' se o número de bits a '1' for par e envia um bit a '0' se o número de bits a '1' for ímpar
   private static int parityCheck(int bit_counter) {
      return (bit_counter % 2 == 0) ? DATA_MASK : 0;
   }

   // Simula um clock dado pela máscara.
   private static void clockPulse() {
      HAL.setBits(CLOCK_MASK);
      //Time.sleep(1);
      HAL.clrBits(CLOCK_MASK);
   }

}
```

E. Código Java da classe *LCD*

```
package edu.isel.lic.peripherals.lcd;

import edu.isel.lic.link.SerialEmitter;
import isel.leic.utils.*;
import edu.isel.lic.link.HAL;

public class LCD // Escreve no LCD usando a interface a 4 bits.
{
  public static final int LINES = 2, COLS = 16;        // Dimensão do display.
  public static final int SERIAL_DATA_SIZE = 5;        // Dimensão da data a enviar para SDX
  public static final int RS_MASK = 0x10;              // Máscara para selecionar entre Data/Command
  public static final int PARALLEL_ENABLE_MASK = 0x20;  // Mascara do bit de enable em modo paralelo
  private static final int DDRAM_LINE = 0x40;

  private static final int[]
      spaceship_pattern = {
      0B11110,
      0B11000,
      0B11100,
      0B11111,
      0B11100,
      0B11000,
      0B11110,
      0B00000},

      invader_pattern = {
      0B11111,
      0B11111,
      0B10101,
      0B11111,
      0B11111,
      0B10001,
      0B10001,
      0B00000};

  public static final CustomCharacter spaceship = new CustomCharacter(spaceship_pattern);
  public static final CustomCharacter invader = new CustomCharacter(invader_pattern);

  // Define se a interface com o LCD é série ou paralela
  private static final boolean SERIAL_INTERFACE = true;

  // Escreve um nibble de comando/dados no LCD em paralelo
  private static void writeNibbleParallel (boolean rs, int data) {
    HAL.setBits(rs ? RS_MASK : 0);                  //rs on/off
    HAL.setBits(data);                              //data on/off
    HAL.setBits(PARALLEL_ENABLE_MASK);              //enable on
    HAL.clrBits(PARALLEL_ENABLE_MASK);              //enable off
    HAL.clrBits(HAL.MAX_BITS);                      //clear bits
  }

  // Escreve um nibble de comando/dados no LCD em série
  private static void writeNibbleSerial (boolean rs, int data) {
    data = (rs) ? (data<<1)+0x01 : data<<1;
    SerialEmitter.send(SerialEmitter.Destination.SLCD, SERIAL_DATA_SIZE, data);
  }

  // Escreve um nibble de comando/dados no LCD
  private static void writeNibble (boolean rs, int data) {
    if(SERIAL_INTERFACE)
      writeNibbleSerial(rs, data);
    else
      writeNibbleParallel(rs, data);
  }

  // Escreve um byte de comando/dados no LCD
  private static void writeByte (boolean rs, int data) {
    int lowerBits = data & 0x0F, higherBits = data>>>4;
    writeNibble(rs, higherBits); writeNibble(rs, lowerBits);
  }

  // Escreve um comando no LCD
  public static void writeCMD (int data) { writeByte (false, data); }

  // Escreve um dado no LCD
  public static void writeDATA (int data) { writeByte (true, data);
}

  // Escreve um caráter na posição corrente.
  public static void write (char c) { writeDATA (c); }

  // Escreve uma string na posição corrente.
  public static void write (String txt) {
    for (int i = 0; i < txt.length(); ++i)
      write(txt.charAt(i));
  }

  // Escreve um caráter customizado na posição corrente.
  public static void write (CustomCharacter custom_char) {
    writeDATA((char)(custom_char.getAddr()));
  }

  // Envia comando para posicionar cursor ('lin':0..LINES-1 , 'col':0..COLS-1)
  public static void cursor (int lin, int col) {
    writeCMD(LCDCode.set_ddram_address(((lin * DDRAM_LINE)) | col));
  }

  // Envia comando para limpar o ecrã e posicionar o cursor em (0,0)
  public static void clear () { writeCMD (LCDCode.clear_display());
}

  // Envia a sequência de iniciação para comunicação a 4 bits.
  public static void init () {
    //init1 -> 0011 -> 0110 -> 110
    Time.sleep(20);
    writeNibble(false, 0x03);
    //init2 -> 0011 -> 0110 -> 110
    Time.sleep(5);
    writeNibble(false, 0x03);
    //init3 -> 0011-> 0110 -> 110
    Time.sleep(1);
    writeNibble(false, 0x03);
    //set 4bit mode -> 0010 -> 0100 -> 100
    writeNibble(false, 0x02);
    //number of display lines and character font -> 0010 -> 0100 -> 100 & 1000 -> 10000
    writeCMD(LCDCode.function_set(false, true, false));
    //display off -> 0000 -> 0 & 1000 -> 10000
    writeCMD(LCDCode.display_control(false, false, false));
    //display clear -> 0000 -> 0 & 0001 -> 0010 -> 10
    writeCMD(LCDCode.clear_display());
    //cursor direction and display shift mode -> 0000 -> 0 & 0110 -> 1100
    writeCMD(LCDCode.entry_mode_set(true, false));
    //display on (entire display, cursor on, cursor blinking on) -> 0000 -> 0 & 1111 -> 11110
    writeCMD(LCDCode.display_control(true, false, false));
    //Custom Characters
    CustomCharacter.add(spaceship);
    CustomCharacter.add(invader);

    clear();
  }
}
```

F. Código Java da classe *LCDCode*

```
package edu.isel.lic.peripherals.lcd;

import edu.isel.lic.link.HAL;

public class LCDCode // Baseado no pdf QuickReference do
{
  private static int value = 0; // valor hexadecimal do código
correspondente
  private static final int
    CLEAR_DISPLAY = 0x01,
    RETURN_HOME = 0x02,
    ENTRY_MODE_SET = 0x04,
    DISPLAY_CONTROL = 0x08,
    CURSOR_DISPLAY_SHIFT = 0x10,
    FUNCTION_SET = 0x20,
    SET_CGRAM_ADDR = 0x40,
    SET_DDRAM_ADDR = 0x80;

  // Para efeitos de teste - needs work
  public static void main (String[] args) {
    HAL.init();
    LCD.init();
  }

  // Envia comando para limpar o ecrã e posicionar o cursor em
(0,0)
  public static int clear_display() { return CLEAR_DISPLAY; }

  // Retorna o ecrã e o cursor para a posição original (endereço 0)
  public static int return_home() { return RETURN_HOME; }

  /**
   * Define a direção do movimento do cursor e especifica a
deslocação do ecrã
   * @param cursor_move_to_right - true: cursor move-se para a
direita; false - cursor move-se para a esquerda
   * @param cursor_follows_display_shift - true: ecrã desloca-se
com o cursor; false: ecrã não se desloca com o cursor
   * @return - valor hexadecimal do código correspondente
   */
  public static int entry_mode_set(boolean cursor_move_to_right,
boolean cursor_follows_display_shift) {
    return value =
        ENTRY_MODE_SET +
        ((cursor_move_to_right) ? 0x02 : 0) +
        ((cursor_follows_display_shift) ? 0x01 : 0);
  }

  /**
   * Liga/desliga o ecrã, liga/desliga cursor e liga/desliga cursor a
piscar
   * @param display_on - true: liga o ecrã; false - desliga o ecrã
   * @param cursor_on - true: liga o cursor; false - desliga o cursor
   * @param cursor_blink - true: liga o modo cursor a piscar; false -
desliga o modo cursor a piscar
   * @return - valor hexadecimal do código correspondente
   */
  public static int display_control (boolean display_on, boolean
cursor_on, boolean cursor_blink) {
    return value =
        DISPLAY_CONTROL +
        ((display_on) ? 0x04 : 0) +
        ((cursor_on) ? 0x02 : 0) +
        ((cursor_blink) ? 0x01 : 0);
  }

  /**
   * Define o modo de deslocação do cursor e do ecrã
   * @param display_shift - true: o ecrã desloca-se; false: o cursor
desloca-se
   * @param shift_to_right - true: deslocamento para a direita; false
- deslocamento para a esquerda
   * @return - valor hexadecimal do código correspondente
   */
  public static int cursor_or_display_shift (boolean display_shift,
boolean shift_to_right) {
    return value =
        CURSOR_DISPLAY_SHIFT +
        ((display_shift) ? 0x08 : 0) +
        ((shift_to_right) ? 0x04 : 0);
  }

  /**
   * Define a dimensão da interface, número de linhas a mostrar e a
fonte das letras
   * @param interface_data_length_8bits - true: interface a 8 bits;
false: interface a 4 bits
   * @param number_of_display_lines_2 - true: 2 linhas no ecrã;
false: 1 linha no ecrã
   * @param character_font_5x10 - true: fonte do tipo 5x10; false:
fonte do tipo 5x8
   * @return - valor hexadecimal do código correspondente
   */
  public static int function_set (boolean
interface_data_length_8bits, boolean number_of_display_lines_2,
boolean character_font_5x10) {
    return value =
        FUNCTION_SET +
        ((interface_data_length_8bits) ? 0x10 : 0) +
        ((number_of_display_lines_2) ? 0x08 : 0) +
        ((character_font_5x10) ? 0x04 : 0);
  }

  /**
   * Defina endereço para o módulo CGRAM
   * @param addr - endereço (6 bits)
   * @return - valor hexadecimal do código correspondente
   */
  public static int set_cgram_address (int addr) {
    return value =
        SET_CGRAM_ADDR + addr;
  }

  /**
   * Define endereço para o módulo DDRAM
   * @param addr - endereço (7 bits)
   * @return - valor hexadecimal do código correspondente
   */
  public static int set_ddram_address (int addr) {
    return value =
        SET_DDRAM_ADDR + addr;
  }
}
```

G. Código Java da classe *CustomCharacter*

```
package edu.isel.lic.peripherals.lcd;

import java.util.ArrayList;

public class CustomCharacter {

  ArrayList<Integer> char_pattern = new ArrayList<Integer>();

  private final int ddram_addr, fontHeight;
  private static int numOfChars = 0;
  private static final int MAX_5x8_numOfChars = 8, MAX_5x10_numOfChars = 4;

  public CustomCharacter (int[] pattern_array) {
    for (int index : pattern_array)
        this.char_pattern.add(index);

    ddram_addr = numOfChars; // Font=5x8 -> cgram_addr={0,1,2,3,4,5,6,7} | Font=5x10 -> cgram_addr={0,1,2,3}
    fontHeight = pattern_array.length; // Font=5x8 -> fontHeight=8 | Font=5x10 -> fontHeight=10
    numOfChars++;

    // Algoritmo para dar ciclo aos endereços na CGRAM caso encher até ao fim o espaço disponível
    if (fontHeight == 8)
      numOfChars = (numOfChars == MAX_5x8_numOfChars) ? 0 : numOfChars++;
    else
      numOfChars = (numOfChars == MAX_5x10_numOfChars) ? 0 : numOfChars++;
    }

  public static void add (CustomCharacter char_object) {

    for (int i = 0; i < char_object.fontHeight; ++i) {
      LCD.writeCMD(LCDCode.set_cgram_address(char_object.ddram_addr * char_object.fontHeight + i));
      LCD.writeDATA(char_object.char_pattern.get(i));
    }
    LCD.writeCMD(LCDCode.set_ddram_address(0));
  }

  public int getAddr () {
    return ddram_addr;
  }
}
```

H. Código Java da classe *SoundGenerator*

```
package edu.isel.lic.link.sound;

import edu.isel.lic.link.SerialEmitter;

public class SoundGenerator { // Controla o Sound Generator.

  public static final int SERIAL_DATA_SIZE = 4;

  // Envia comando para reproduzir um som, com a identificação deste
  public static void play(SGCode.Sound sound) {
    SerialEmitter.send(SerialEmitter.Destination.SSC, SERIAL_DATA_SIZE, SGCode.set_sound(sound));
    SerialEmitter.send(SerialEmitter.Destination.SSC, SERIAL_DATA_SIZE, SGCode.play());
  }

  // Envia comando para parar o som
  public static void stop() {
    SerialEmitter.send(SerialEmitter.Destination.SSC, SERIAL_DATA_SIZE, SGCode.stop());
  }

  // Envia comando para definir o volume do som
  public static void setVolume(SGCode.Volume volume) {
    SerialEmitter.send(SerialEmitter.Destination.SSC, SERIAL_DATA_SIZE, SGCode.set_volume(volume));
  }

  // Inicia a classe, estabelecendo os valores iniciais.
  public static void init() {
    stop();
  }
}
```

I. Código Java da classe *SGCode*

```
package edu.isel.lic.link.sound;

public class SGCode
{
  private static int value = 0;

  private static final int
    STOP = 0x0,
    PLAY = 0x1,
    SET_SOUND = 0x2,
    SET_VOLUME = 0x3;

  public enum Sound {
    GAME_OVER(0x1),
    SOUND1(0x0),
    SOUND2(0x2),
    SOUND3(0x3);

    private final int sound;

    Sound(int sound) { this.sound = sound; }

    private int getValue() { return sound; }
  }

  public enum Volume {
    MUTE(0x0),
    LOW(0x1),
    MED(0X2),
    HIGH(0x3);

    private final int volume;

    Volume(int volume) { this.volume = volume; }

    private int getValue() { return volume; }
  }

  public static int stop() {
    return STOP;
  }

  public static int play() {
    return PLAY;
  }

  public static int set_sound (Sound sound) {
    return value =
        SET_SOUND +
        (sound.getValue() << 2);
  }

  public static int set_volume (Volume volume) {

    return value =
        SET_VOLUME +
        (volume.getValue() << 2);
  }

}
```

J. Código Java da classe *M*

```
package edu.isel.lic.link;

public class M
{
  public static final int M_BUTTON = 0x80;

  // Verifica se o botão de manutenção está ligado
  public static boolean checkButton () {
    return HAL.isBit(M_BUTTON);
  }
}
```

K. Código Java da classe *COIN*

```
package edu.isel.lic.link;

import isel.leic.UsbPort;

public class COIN
{
  public static final int COIN_BUTTON = 0x40, accept_MASK = 0x40;

  // Retorna de imediato se o botão de COIN foi pressionado
  public static boolean buttonPress ()
  {
    boolean buttonPress = false;

    if (HAL.isBit(COIN_BUTTON)) {
      buttonPress = true;
      HAL.setBits(accept_MASK);
      HAL.clrBits(accept_MASK);
    }

    return buttonPress;
  }

  // Inicia a classe
  public static void init() {
    HAL.clrBits(accept_MASK);
  }
}
```

L. Código Java da classe *TUI*

```
package edu.isel.lic.game;

import edu.isel.lic.link.HAL;
import edu.isel.lic.link.SerialEmitter;
import edu.isel.lic.peripherals.KBD;
import edu.isel.lic.peripherals.lcd.CustomCharacter;
import edu.isel.lic.peripherals.lcd.LCD;
import edu.isel.lic.peripherals.lcd.LCDCode;
import isel.leic.utils.Time;
import java.util.Scanner;
import java.lang.Math;

public class TUI
{
   public static final int LINES = LCD.LINES, COLS = LCD.COLS;
   private static final char[][] lcdState = new char[LINES][COLS];
   public static final char NONE = ' ', CUSTOM_CHAR = '^';
   public enum Format { ALIGN_CENTER, ALIGN_RIGHT,
ALIGN_LEFT, SHIFT_RIGHT, SHIFT_LEFT }
   public enum Direction { RIGHT, LEFT }
   public enum Control { DISPLAY, CURSOR }

   // Imprime no LCD o que está em lcdState
   public static void print (int lin, int startCol, int endCol) {
      String str = readArray (lin, startCol, endCol);
      LCD.cursor(lin, startCol);
      LCD.write(str);
   }

   // Imprime no LCD a String recebida e atualiza lcdState
   public static void print (int lin, int col, String str) {
      LCD.cursor(lin, col);
      LCD.write(str);
      writeArray(lin, col, str);
   }

   // Imprime no LCD o char recebida e atualiza lcdState
   public static void print (int lin, int col, char c) {
      LCD.cursor(lin, col);
      LCD.write(c);
      writeArray(lin, col, c);
   }

   // Imprime no LCD o char customizado recebido e atualiza
lcdState
   public static void print (int lin, int col, CustomCharacter
custom_char) {
      LCD.cursor(lin, col);
      LCD.write(custom_char);
      writeArray(lin, col, CUSTOM_CHAR);
   }

   // Apaga uma String no LCD
   public static void clear (int lin, int startCol, int endCol) {
      for (int col = startCol; col <= endCol; ++col)
         lcdState[lin][col] = NONE;
      print (lin, startCol, endCol);
      LCD.cursor(lin, startCol);
   }

   // Apaga um char no LCD
   public static void clear (int lin, int col) {
      lcdState[lin][col] = NONE;
      print (lin, col, col);
      LCD.cursor(lin, col);
   }

   // Apaga uma linha completa no LCD
   public static void clearLine (int lin) { clear (lin, 0, LCD.COLS - 1);
}

   // Apaga tudo no LCD
   public static void clearAll() {
      clear (0, 0, LCD.COLS - 1);
      clear (1, 0, LCD.COLS - 1);
      LCD.clear();
   }

   // Escreve uma String para o lcdState
   private static void writeArray (int lin, int startCol, String str) {
      for (int col = startCol, strIndex = 0; col < LCD.COLS &&
strIndex < str.length(); ++col, ++strIndex)
         writeArray(lin, col, str.charAt(strIndex));
   }

   // Escreve um char para o lcdState
   private static void writeArray (int lin, int col, char c) {
lcdState[lin][col] = c; }

   // Lê uma String do lcdState
   public static String readArray (int lin, int startCol, int endCol) {
      int length = endCol - startCol;
      StringBuilder str = new StringBuilder(length);

      for (int col = startCol; col <= endCol; ++col)
         str.append(lcdState[lin][col]);

      return str.toString();
   }

   // Lê um char do lcdState
   public static char readArray (int lin, int col) { return
lcdState[lin][col]; }

   // Move posição de conteúdo no LCD
   public static void moveText (int linInitial, int linFinal, int
startColInitial, int endColInitial, int startColFinal) {
      String str = readArray (linInitial, startColInitial, endColInitial);
      clear (linInitial, startColInitial, endColInitial);

      int endColFinal = startColFinal + str.length() - 1;
      writeArray (linFinal, startColFinal, str);

      if (invalidIndex (0, endColFinal))
         endColFinal = LCD.COLS - 1;

      if (invalidIndex (startColFinal, LCD.COLS - 1))
         startColFinal = 0;

      print(linFinal, startColFinal, endColFinal);
   }

   // Formata o texto presente no lcdState
   public static void formatText (int lin, int startColInitial, int
endColInitial, Format format) {
      int startColFinal = selectText ( startColInitial, endColInitial,
format);
      moveText(lin, lin, startColInitial, endColInitial, startColFinal);
   }

   // Cria e formata esse texto no LCD
   public static void create_formatText (int lin, Format format,
String str) {
      int startColInitial = 0;
      int endColInitial = startColInitial + str.length();
      int startColFinal = selectText( startColInitial, endColInitial,
format);

      print (lin, startColFinal, str);
   }

   // Seleciona texto e retorna as coordenadas finais dadas pelo
formato escolhido
   public static int selectText (int startColInitial, int endColInitial,
Format format) {
      int length = endColInitial - startColInitial + 1, startColFinal =
startColInitial, increment = 0;

      switch (format) {
         case ALIGN_CENTER:
            startColFinal = LCD.COLS/2 - length/2;
            break;
```

```
            case ALIGN_RIGHT:
               startColFinal = LCD.COLS - length + 1;
               break;
            case ALIGN_LEFT:
               startColFinal = 0;
               break;
            case SHIFT_RIGHT:
               increment = 1;
               break;
            case SHIFT_LEFT:
               increment = -1;
               break;
            default:
               System.err.println("No such direction.");
               startColFinal = -1;
               break;
         }

         if(format == Format.SHIFT_RIGHT || format ==
Format.SHIFT_LEFT)
            startColFinal += increment;

         return startColFinal;
      }

      // Cria um meno com título e duas escolhas
      public static void menu (Format format, String title, String
choiceA_text, String choiceB_text) {
         clearAll();
         create_formatText(0, format, title);
         create_formatText(1, Format.ALIGN_LEFT, choiceA_text);
         create_formatText(1, Format.ALIGN_RIGHT, choiceB_text);
      }

      // Verifica se é um índice válido de lcdState
      private static boolean invalidIndex(int startCol, int endCol) {
         boolean invalidIndex = true;
         if (startCol >= 0 && endCol < LCD.COLS)
            invalidIndex = false;
         return invalidIndex;
      }

      // Ativa/desativa o cursor
      public static void cursorControl (boolean cursorOn, boolean
cursorBlink) {
         LCD.writeCMD(LCDCode.display_control(true, cursorOn,
cursorBlink));
      }

      // Move o cursor para a posição indicada pelos parâmetros
recebidos
      public static void moveCursor (int lin, int col) {
         LCD.cursor(lin, col);
      }

      // Desloca o cursor/ecrã uma vez para a direção indicada pelos
parâmetros recebidos
      public static void shiftControl (Control ctrl, Direction dir) {
         boolean shift_to_right = dir == Direction.RIGHT;
         boolean display_shift = ctrl == Control.DISPLAY;


LCD.writeCMD(LCDCode.cursor_or_display_shift(display_shift,shift
_to_right));
      }

      // Inicia esta classe
      public static void init () {
         clearAll();
      }
   }
```

M. Código Java da classe *FileAccess*

```
package edu.isel.lic.game;

import java.io.*;
import java.util.Scanner;

public class FileAccess
{
  public static void writeFileText(String[] text ,String file_name)
  {
    PrintWriter fileToWrite = writeTextVar(file_name);
    for (String s : text) {
      fileToWrite.println(s);
    }
    fileToWrite.flush();
    fileToWrite.close();
  }

  public static String[] readFile(int lines, String file_name)
  {
    String[] array = new String[lines];
    Scanner fileToRead = readTextVar(file_name);

    for (int i = 0; i < lines ; ++i) {
      if(fileToRead.hasNextLine()) {
        array[i] = fileToRead.nextLine();
      }else{
        array[i] = "" ;
      }
    }
    return array;
  }


  // Abrir o FileReader e Scanner para um ficheiro inputFile ou
outputFile.
  public static Scanner readTextVar(String file_name){

    file_name = fileName(file_name);


    Scanner readText = null;
    try {
      readText = new Scanner(new FileReader(file_name));
    } catch (FileNotFoundException e) {
      e.printStackTrace();
      System.out.println("File missing: " + file_name);
      System.exit(-1);
      //createFile(file_name);


    }
    return readText;
  }

  // Abrir o FileWriter e PrintWriter para um ficheiro inputFile ou
outputFile.
  public static PrintWriter writeTextVar(String file_name){

    file_name = fileName(file_name);

    PrintWriter writeText = null;
    try {
      writeText = new PrintWriter(new FileWriter(file_name));
    } catch (IOException e) {
      e.printStackTrace();
      System.out.println("Unable to save: "+ file_name);
      System.exit(-2);
    }

    return writeText;
  }

  // Determina a partir de uma String em qual ficheiro se vai
ler/escrever.
  private static String fileName (String file_name){
    file_name = file_name+".txt";
    return file_name ;
  }
}
```

N. Código Java da classe *Scores*

```
package edu.isel.lic.game;

public class Scores {
  public static final int PLACEMENTS = 10 ;
  private static String[] playerName = new String[PLACEMENTS];
  private static int[] playerScore = new int[PLACEMENTS];

  public static int getPlayerScore(int pos){
    return playerScore[pos - 1];
  }

  public static String getPlayerName (int pos){
    if (playerName[pos-1]==null){
      return "";
    }else {
      return playerName[pos - 1];
    }
  }

  public static void loadScores (){

    String[] scoresArray =
FileAccess.readFile(PLACEMENTS,"ScoreBoard");
    for( int i = 0 ; i<10 ; ++i ){
      String score = "";
      String name = "" ;
      int t = 0 ;

      if(scoresArray[i].length()==0){ return; }

      while(scoresArray[i].charAt(t) != ';'){
        score += scoresArray[i].charAt(t++);
      }

      while(++t < scoresArray[i].length()){
        name += scoresArray[i].charAt(t) ;
      }

      playerScore[i] =Integer.parseInt(score) ;
      playerName[i] = name;

    }
  }

  public static void archiveScores(){
    String[] scores = new String[PLACEMENTS];

    for (int i = 0 ; i < PLACEMENTS ; ++i){
      if(playerName[i]==null){
        scores[i]="";
      }else {
        scores[i] = "" + playerScore[i] + ";" + playerName[i];
      }
    }
    FileAccess.writeFileText(scores,"ScoreBoard");
  }
  public static void saveScore(int score , String name ){
    int position = findNewPlayerPosition(score);
    if(position<PLACEMENTS) {
      for (int i = PLACEMENTS - 1; i > position; --i) {
        playerScore[i] = playerScore[i - 1];
        playerName[i] = playerName[i - 1];
      }
      playerScore[position]=score;
      playerName[position]=name;
    }
  }

  private static int findNewPlayerPosition(int score){
    int position=PLACEMENTS;
    for(int i = PLACEMENTS-1 ; score>playerScore[i] ; --i){
      --position ;
      if(i==0) break; // idkw but i cant make it work without this
TODO: repair this.
    }
    return position;

  }

  public static int getSize ()
  {
    int size = 0;

    for (int value : playerScore)
    {
      if (value != 0)
        size++;
    }

    return size;
  }
}
```

O. Código Java da classe *Statistics*

```
package edu.isel.lic.game;

public class Statistics {
   private static int total_games = 0, total_coins = 0;
   private static final int total_games_line = 0, total_coins_line = 1;
   private static final int numOfStats = 2;

   public static void loadStats(){
      String[] stats = FileAccess.readFile(numOfStats,"statistics");

      if (stats[total_games_line].length() > 0)
         total_games = Integer.parseInt(stats[total_games_line]);

      if (stats[total_coins_line].length() > 0)
         total_coins = Integer.parseInt(stats[total_coins_line]);
   }

   public static void saveStats(int games, int coins){
      total_games=games;
      total_coins=coins;
      String[] stats = new String[numOfStats];
      stats[0]=""+total_games;
      stats[1]=""+total_coins;
      FileAccess.writeFileText(stats , "statistics");

   }

   public static int totalGames(){
      return total_games;
   }

   public static int totalCoins(){
      return total_coins;
   }
}
```

P. Código Java da classe *APP*

```
package edu.isel.lic.game;

import edu.isel.lic.link.COIN;
import edu.isel.lic.link.HAL;
import edu.isel.lic.link.M;
import edu.isel.lic.link.SerialEmitter;
import edu.isel.lic.link.sound.SGCode;
import edu.isel.lic.link.sound.SoundGenerator;
import edu.isel.lic.peripherals.KBD;
import edu.isel.lic.peripherals.lcd.LCD;
import edu.isel.lic.peripherals.lcd.LCDCode;
import isel.leic.utils.Time;

public class APP
{
   public static final String INVADERS = "0123456789";
//Invasores a utilizar.
   public static int current_coins, total_coins;
//current_coins: moedas que o utilizador introduziu; total_coins: moedas que a máquina tem no total
   public static int current_score = 0, total_games;
//current_score: pontuação do jogador; gameCounter = jogos realizados na máquina no total
   public static boolean running = true, saveStats = true;
//jogo a correr

   /**
    * Main do edu.isel.lic.game.APP
    * Inicializa a classe, entra num ciclo controlado pelo booleano running
    * Tem 3 fases: menu, game e score;
    * No fim de cada jogo incrementa ao número de jogos realizados: gameCounter
    * @param args - Sem args iniciais
    */
   public static void main (String[] args) {
      init();
      while (running) {
         menu();
         game();
         if (saveStats)
            score();
      }
   }

   /**
    * Inicializa esta classe
    */
   private static void init()
   {
      HAL.init();
      SerialEmitter.init();
      KBD.init();
      LCD.init();
      SoundGenerator.init();
      COIN.init();
      TUI.init();
      Scores.loadScores();
      Statistics.loadStats();
      total_coins = Statistics.totalCoins();
      total_games = Statistics.totalGames();
   }

   private static void menu() {
      char key;
      boolean hasCoin = (current_coins > 0), newGame = false, showNewScore = true, showMenu = true;
      long view_score_timeout = 3000, starting_time = Time.getTimeInMillis();
      int score_tracker = 1;

      do {
         key = KBD.getKey();

         if (showMenu) {
            TUI.clearAll();
            TUI.create_formatText(0, TUI.Format.ALIGN_CENTER, "Space Invaders");
            showMenu = false;
         }

         // Mostrar pontuações
         long current_time = Time.getTimeInMillis();
         if (showNewScore || key != KBD.NONE) {

            TUI.clearLine(1);
            if (Scores.getSize() > 0 && !Scores.getPlayerName(score_tracker).equals("")) {
               TUI.create_formatText(1, TUI.Format.ALIGN_LEFT, "" + score_tracker + ":");
               TUI.create_formatText(1, TUI.Format.ALIGN_CENTER, "" + Scores.getPlayerName(score_tracker) + "-" + Scores.getPlayerScore(score_tracker));
            } else {
               TUI.create_formatText(1, TUI.Format.ALIGN_CENTER, "No games played.");
            }

            score_tracker++;
            if (score_tracker > Scores.getSize())
               score_tracker = 1;

            starting_time = Time.getTimeInMillis();
         }
         showNewScore = (current_time - starting_time >= view_score_timeout);

         // Contabilizar moedas
         if (COIN.buttonPress()) {
            ++current_coins;
            TUI.clearLine(1);
            TUI.print(1, 0, " Game ");
            TUI.create_formatText(1, TUI.Format.ALIGN_RIGHT, "$" + current_coins);
            TUI.print(1, 6, LCD.spaceship);
            TUI.print(1, 9, LCD.invader);
            TUI.print(1, 11, LCD.invader);
            hasCoin = true;
         }

         // Verificar condição de início de jogo
         if (hasCoin && key == '*') {
            newGame = true;
            current_coins--;
            total_coins++;
         }

         // Menu de manutenção
         boolean showMaintenanceMenu = true;
         while (M.checkButton() && showNewScore) {
            key = KBD.getKey();

            if (showMaintenanceMenu) {
               TUI.menu(TUI.Format.ALIGN_CENTER, "On Maintenance", "*-Count", "#-shutD");
               showMaintenanceMenu = false;
            }

            if (key == '*') {
               TUI.clearAll();
               TUI.print(0, 0, "Games:" + Statistics.totalGames());
               TUI.print(1, 0, "Coins:" + Statistics.totalCoins());
               key = KBD.waitKey(5000);
               if (key == '#') {
                  TUI.menu(TUI.Format.ALIGN_CENTER, "Clear counters", "5-Yes", "other-No");
                  key = KBD.waitKey(5000);
                  if (key == '5') {
                     total_coins = 0;
                     total_games = 0;
                     Statistics.saveStats (total_games, total_coins);
                  }
```

```
                showMaintenanceMenu = true;
              }
            }
            else if (key == '#') {
              TUI.menu(TUI.Format.ALIGN_CENTER, "Shutdown?",
"5-Yes", "other-No");
              key = KBD.waitKey(5000);
              if (key == '5') {

LCD.writeCMD(LCDCode.display_control(false,true,true));
                Scores.archiveScores();
                Statistics.saveStats (total_games, total_coins);
                System.exit(0);
              }
              showMaintenanceMenu = true;
            }
            else if (key != KBD.NONE) {
              newGame = true;
              saveStats = false;
            }
            showMenu = true;
          }
        } while (!hasCoin || !newGame);
    }

    public static void game ()
    {
      char key, number_key = KBD.NONE;
      int invader_tracker = 0, max_invaders = 14;
      long spawnInvader_timeout = 1500, starting_time =
Time.getTimeInMillis();
      boolean spawnInvader = true, invader_killed, gameOver =
false;
      double time_multiplier = 0.005;

      current_score = 0;
      TUI.clearAll();
      TUI.print (0, 0, "]");
      TUI.print(0, 1, LCD.spaceship);
      TUI.print (1, 0, "Score:" + current_score);

      do {
        key = KBD.getKey();

        // Faz nascer um invader se passou o tempo da condição
        long current_time = Time.getTimeInMillis();
        if (spawnInvader) {
          spawnInvader();
          invader_tracker++;
          starting_time = Time.getTimeInMillis();
        }
        spawnInvader = (current_time - starting_time >=
spawnInvader_timeout);

        // Atualiza o número alvo à esquerda da nave
        if (key != KBD.NONE && key != '*' && key != '#') {
          number_key = key;
          TUI.print(0, 0, "" + number_key);
        }

        // Verifica se invader foi morto ou não e atualiza o score
        if (key == '*' && number_key != KBD.NONE &&
invader_tracker > 0) {
          invader_killed = killInvader(number_key);
          number_key = KBD.NONE;
          if (invader_killed) {
            invader_tracker--;
            updateScore();
            if (spawnInvader_timeout > 200)
              spawnInvader_timeout -= spawnInvader_timeout *
time_multiplier;
          }
        }

        // Condição de fim de jogo
        if (invader_tracker > max_invaders)
          gameOver = true;
      } while (!gameOver);

      TUI.clearLine (0);
      TUI.create_formatText (0, TUI.Format.ALIGN_CENTER, "Game
Over");
      SoundGenerator.setVolume(SGCode.Volume.MED);
      SoundGenerator.play(SGCode.Sound.GAME_OVER);
      KBD.waitKey(3000);
      SoundGenerator.stop();
    }

    /**
     * Método para fazer nascer um invasor
     * new_invader - caráter do novo invasor (é gerado um número de
0 a 9 para selecionar na string INVADERS a sua posição
correspondente)
     */
    private static void spawnInvader () {
      TUI.formatText (0, 2, TUI.COLS - 1, TUI.Format.SHIFT_LEFT);
      TUI.print(0, 1, LCD.spaceship);
      char new_invader =
INVADERS.charAt((int)(Math.random()*10));
      TUI.create_formatText(0, TUI.Format.ALIGN_RIGHT, "" +
new_invader);
    }

    /**
     * Método para verificar se é ou não morto o invasor
     * invader_killed - true: invasor foi morto; false: invasor não foi
morto
     * invaders - é lido os invaders que estão no ecrã, elimina-se os
espaços (edu.isel.lic.peripherals.KBD.NONE) e guarda-se o caráter da
posição 0
     * lcd_col - coluna em que o invasor mais perto da nave se encontra
     * invader - carater que representa o invasor mais perto da nave
     * @param key - number_key enviado do método game(), isto é, a
tecla com um número
     * @return - Retorna true se o invasor foi morto e false se não foi
morto
     */
    private static boolean killInvader (char key)
    {
      boolean invader_killed = false;

      String invaders = TUI.readArray (0,  2, TUI.COLS - 1);
      invaders = invaders.replaceAll("\\s", "");
      int target_invader_col = TUI.COLS - invaders.length();
      char invader = invaders.charAt(0);
      if (key == invader) {
        TUI.clear (0, target_invader_col);
        invader_killed = true;
      }

      return invader_killed;
    }

    /**
     * Método que atualiza a pontuação nas variáveis globais e no lcd
     */
    private static void updateScore() {
      current_score++;
      TUI.clear (1, 6, TUI.COLS - 1);
      TUI.print (1, 6, "" + current_score);
    }

    /**
     * Inicializa a função de guardar a pontuação do jogador com o seu
nome correspondente
     * key - tecla pressionada
     * current_letter - tecla onde o cursor se encontra
     * current_col - (nova) coluna onde o cursor se encontra
     * Este método conta com uma adição: o ciclo das letras dá volta,
isto é, quando se encontra em A posso ir para Z e vice-versa
     */
    private static void score ()
    {
      char key = KBD.NONE;
      char current_letter = 'A';
```

```
int current_col = 5;

TUI.clearLine (0);
TUI.print (0, 0, "Name:" + current_letter);
TUI.cursorControl(true,true);
TUI.moveCursor(0, 5);
while (key != '5')
{
  key = KBD.waitKey(100);

  switch (key)
  {
    case '4':
      if (current_col > 5)
        current_col--;
      else
        current_col = 5;

      TUI.moveCursor(0, current_col);
      break;
    case '6':
      if (current_col < TUI.COLS - 1)
        current_col++;
      else
        current_col = TUI.COLS - 1;
      current_letter = TUI.readArray (0, current_col);
      if (current_letter < 'A' || current_letter > 'Z')
        TUI.print(0, current_col, "" + 'A');
      TUI.moveCursor(0, current_col);
      break;
    case '2':
      current_letter = TUI.readArray (0, current_col);
      if (current_letter == 'Z')
        current_letter = 'A';
      else
        current_letter++;
      TUI.print (0, current_col, "" + current_letter);
      TUI.moveCursor(0 , current_col);
      break;
    case '8':
      current_letter = TUI.readArray (0, current_col);
      if (current_letter == 'A')
        current_letter = 'Z';
      else
        current_letter--;
      TUI.print (0, current_col, "" + current_letter);
      TUI.moveCursor(0, current_col);
      break;
    case '*':
      if (current_col > 5)
      {
        TUI.clear (0, current_col);
        current_col--;
      }
      else
        current_col = 5;

      TUI.moveCursor(0, current_col);
      break;
    }
  }
  total_games++;
  String player_name = TUI.readArray (0, 5, TUI.COLS - 1);
  player_name = player_name.replaceAll("\\s", "");
  Scores.saveScore (current_score, player_name);
  Statistics.saveStats(total_games, total_coins);
  TUI.cursorControl(false,false);
 }
}
```