

## 1 Funcionamento da Aplicação

### 1.1 Ecrã Inicial

Ao iniciar a execução do `RouletteGame.kt`, o ecrã inicial é apresentado através do LCD, como representado pela figura 1.

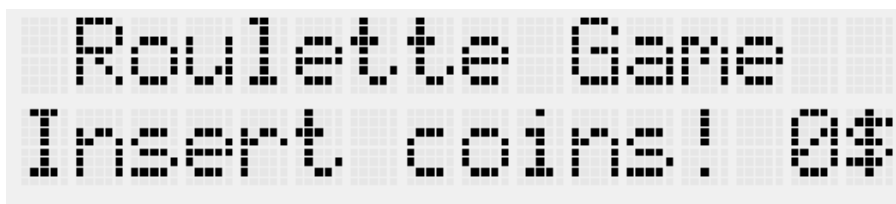


Figura 1: Ecrã Inicial

Este ecrã apresenta:

- O **nome do jogo**, localizado na primeira linha do LCD;
- Um texto rotativo com uma **instrução útil** para o jogador, localizado no canto inferior esquerdo. Se o jogador não inseriu nenhum crédito, a instrução será "Insert coins!", senão a instrução será "Click \* to play!";
- E o **número de créditos**, localizado no canto inferior direito.

Neste momento, o jogador pode:

- **Inserir moedas** para aumentar o número de créditos (ler secção 1.2);
- **Iniciar um novo jogo**, caso haja créditos, através da tecla \* (ler secção 1.3);
- E ativar o **modo manutenção**, através do interruptor SW2 (ler secção 1.5).

### 1.2 Inserção de Moedas

A aplicação permite a inserção de moedas durante o ecrã inicial e o momento de apostar.

Para inserir uma moeda, o jogador começa por escolher que tipo de moeda deseja inserir, recorrendo à tabela 1.

Valor da Moeda	Estado do SW9
2 créditos	Desligado
4 créditos	Ligado

Table 1: Relação entre Moedas e SW9

Após escolher a moeda desejada, esta é inserida através da ativação do SW8. O reconhecimento da inserção da moeda é sinalizado através do LEDR8 ao que o jogador pode desativar o SW8 para finalizar a inserção.

### 1.3 Apostas

Ao começar um novo jogo, é apresentado o ecrã de apostas através do LCD, como representado pela figura 2.

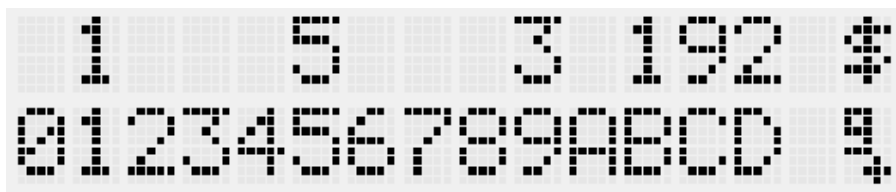


Figura 2: Ecrã de Apostas

Ao entrar neste ecrã, o jogador observa a instrução "Click on these keys to bet!", tendo a lista de teclas apostáveis por baixo dessa instrução. Premindo numa dessas teclas, a instrução é substituída pelo número de créditos apostados em cada tecla, como pode ser visto pela figura 2. O jogador só pode apostar caso tenha créditos e não pode apostar mais de nove créditos numa tecla.

Após cinco segundos sem nenhuma aposta, o ecrã apresenta a instrução "Click # to start or keep betting!", caso o jogador não saiba como terminar as suas apostas. Se não tiver créditos para apostar, aparece a instrução "Click # to start or insert coins!". O jogador poderá inserir moedas durante as apostas (ler secção 1.2).

Ao chegar ao máximo de créditos apostáveis em todas as teclas, não é permitido mais nenhuma aposta e o ecrã apresenta a instrução "Max bets! Click # to start."

No lado direito do ecrã, é apresentado o número de créditos. Devido ao espaço limitado do LCD, se o jogador tiver mais de nove créditos, é apresentado o símbolo 9.

Para iniciar a roleta, o jogador pode premir a tecla #. Quando o fizer, o jogador terá mais cinco segundos para efetuar mais apostas. Um contador decrescente desse tempo é apresentado no canto superior direito e as instruções anteriormente apresentadas deixam de sugerir clicar na tecla #. O mostrador apresenta a animação de roleta.

Quando esses cinco segundos terminam, o LCD passa a apresentar a mensagem "Good luck! Not accepting more bets.", e o mostrador continua com a sua animação de roleta. Após um a cinco segundos, a roleta para e o jogo passa a apresentar o resultado (ler secção 1.4).

## 1.4 Resultado

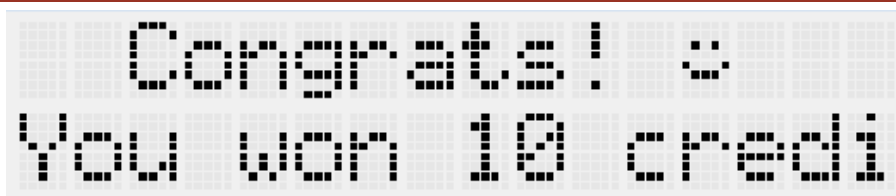
Quando a roleta para, o resultado da roleta é apresentado no lado esquerdo do mostrador e o número de créditos vencidos é apresentado no lado direito, como representado pela figura 3. O número de créditos vencidos é o dobro do que o jogador apostou nessa tecla de forma a que devolva os créditos que apostou e uma bonificação por ter ganho.



Figura 3: Resultado da Roleta no Mostrador

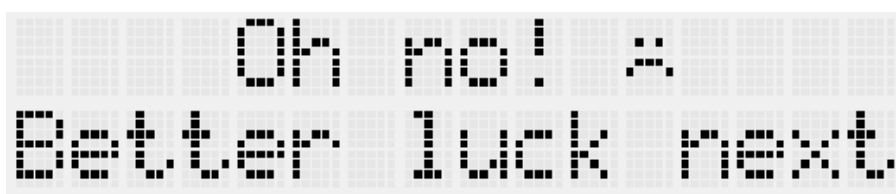
O LCD pode apresentar duas mensagens, dependendo se o utilizador apostou na tecla vencedora.

Caso tenha vencido, apresenta "Congrats! 🎉 You won [n.º de créditos] credits!", representado pela figura 4.



**Figura 4:** Ecrã de Vitória

Caso tenha perdido, apresenta "Oh no! ☹ Better luck next time!", representado pela figura 5.



**Figura 5:** Ecrã de Derrota

Após cinco segundos, os créditos são adicionados e as estatísticas internas são atualizadas, voltando assim ao ecrã inicial (ler secção 1.1).

## 1.5 Modo Manutenção

No ecrã inicial, o administrador pode ativar o interruptor SW2 para entrar no modo manutenção. O menu inicial do modo manutenção é apresentado no LCD, como representado na figura 6.



**Figura 6:** Modo Manutenção

A primeira linha apresenta o nome deste modo, incluindo o símbolo  $\bar{M}$  que aparece em vários momentos do jogo de modo a referenciar que o modo manutenção está ativo.

A segunda linha apresenta um texto rotativo com instruções sobre o que cada tecla faz. Esta informação pode ser consultada pela tabela 2


Tecla	Descrição
*	Inicia um novo jogo sem afetar créditos nem atualizar estatísticas (ler secção 1.5.1).
A	Consulta a contagem de jogos realizados e moedas no cofre do depósito (ler secção 1.5.2).
C	Consulta a contagem de vitórias e créditos vencidos de cada tecla (ler secção 1.5.3).
D	Guarda as estatísticas e desliga a aplicação (ler secção 1.5.4).
#	Ao entrar num dos menus, volta atrás para este ecrã.

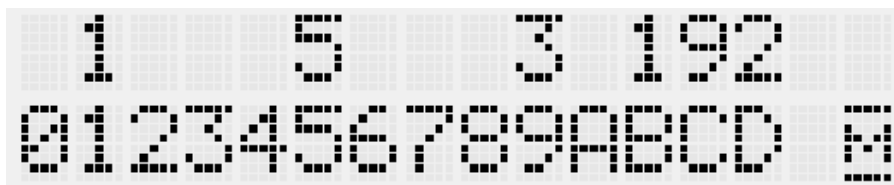
**Table 2:** Teclas no Modo Manutenção

Para sair deste menu, o interruptor SW2 deve ser desativo.

### 1.5.1 Jogo de Teste

Através do modo manutenção, é possível iniciar um jogo de teste que não afete o número de créditos, nem exista limite de quantos créditos se possa usar. Este jogo também não modifica as contagens feitas para as estatísticas.

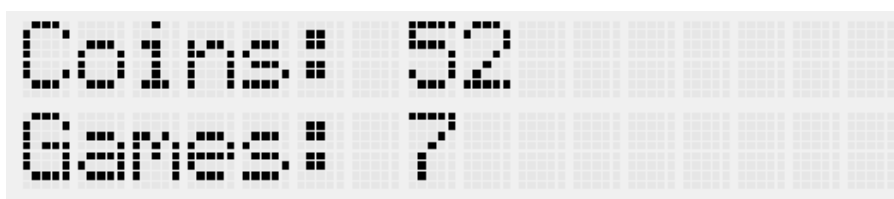
O ecrã de apostas apresenta o símbolo  no canto inferior direito para indicar que é um jogo de teste, como representado na figura 7.



**Figura 7:** Ecrã de Apostas no Modo Manutenção

### 1.5.2 Informação

Através do modo manutenção, é possível consultar a contagem de moedas inseridas e guardadas no cofre do moedeiro e os jogos realizados, como representado na figura 8.



**Figura 8:** Informação

O administrador pode reiniciar estas contagens ao premir a tecla \*.

Para voltar para o menu inicial do modo de manutenção, o administrador deve premir a tecla #.

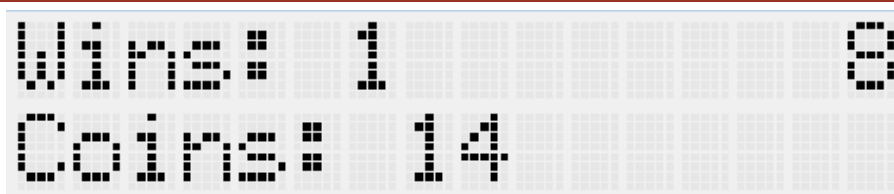
### 1.5.3 Resultados

Ao entrar na consulta de resultados, a aplicação pede ao administrador que prima numa tecla apostável para consultar os resultados desta, como representado na figura 9.



**Figura 9:** Pedido de Tecla

Ao premir numa tecla apostável, o LCD apresenta a contagem de vezes que essa venceu e o total de créditos vencidos nessas vitórias, como representado na figura 10. O canto superior direito apresenta a tecla selecionada.



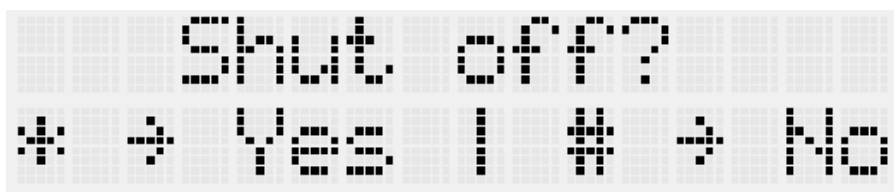
**Figura 10:** Resultados da Tecla 8

O administrador pode seleccionar outras teclas apostáveis sem voltar atrás e pode reiniciar as contagens de todas as teclas ao premir a tecla \*.

Para voltar para o menu inicial do modo de manutenção, o administrador deve premir a tecla #.

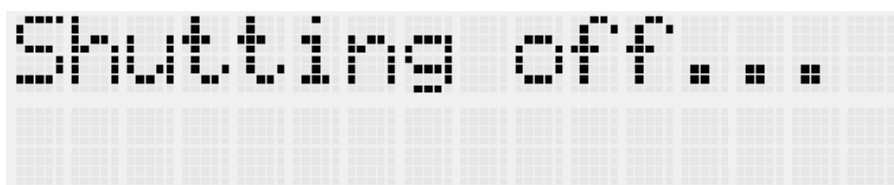
#### 1.5.4 Encerramento

Ao entrar no processo de encerrar a aplicação, o LCD apresenta a confirmação de termino, representado na figura 11. Se o administrador premir a tecla #, volta ao menu inicial do modo manutenção.



**Figura 11:** Confirmação de Encerramento

Ao premir a tecla #, a aplicação guarda as suas estatísticas nos seus ficheiros e termina o seu processo, apresentando uma breve mensagem representada na figura 12.



**Figura 12:** Encerramento

## 1.6 Hardware

Por fim para a conclusão e funcionamento do jogo juntamente com o Hardware foi preciso implementar os componentes *Maintenance* e *CoinAcceptor*. A *Maintenance* é responsável por entrar e sair do modo de manutenção e o *CoinAcceptor* para introduzir as moedas. Ambos os componentes são simples e apenas ligações diretas.

## A VHDL

### A.1 Maintenance

```
library ieee;
use ieee.std_logic_1164.all;

entity Maintenance is
    port(

        -- input
        Maintenance_in: in std_logic;

        -- output
        Maintenance_out: out std_logic

    );
end Maintenance;

architecture logicFunction of Maintenance is begin

    Maintenance_out <= Maintenance_in;

end logicFunction;
```

---

## A.2 CoinAcceptor

```
library ieee;
use ieee.std_logic_1164.all;

entity CoinAcceptor is
  port(

    -- input
    CoinAcceptor_Accept_in: in std_logic;
    CoinAcceptor_Coin_in: in std_logic;
    CoinAcceptor_CoinID_in: in std_logic;

    -- output
    CoinAcceptor_Accept_out: out std_logic;
    CoinAcceptor_Coin_out: out std_logic;
    CoinAcceptor_CoinID_out: out std_logic

  );
end CoinAcceptor;

architecture logicFunction of CoinAcceptor is begin

  CoinAcceptor_Coin_out <= CoinAcceptor_Coin_in;
  CoinAcceptor_CoinID_out <= CoinAcceptor_CoinID_in;
  CoinAcceptor_Accept_out <= CoinAcceptor_Accept_in;

end logicFunction;
```



## B *Kotlin*

### B.1 HAL

#### Algoritmo 1: HAL - Hardware Abstract Layer

```
import isel.leic.UsbPort

object HAL {

    /** ltimo _output_ enviado. */
    var lastOutput = 0

    /** Inicializa o _output_ do UsbPort. */
    fun init() = UsbPort.write(0)

    /** Verifica se os bits da `mask` est o ativos no _input_ do UsbPort. */
    fun isBit(mask: Int) = readBits(mask) == mask

    /** Retorna os bits no _input_ do UsbPort filtrados por `mask`. */
    fun readBits(mask: Int) = UsbPort.read() and mask

    /** Escreve no _output_ do UsbPort `value` filtrado por `mask`. */
    fun writeBits(mask: Int, value: Int) = write((value and mask) or (lastOutput and mask.inv()))

    /** Ativa no _output_ do UsbPort os bits da `mask`. */
    fun setBits(mask: Int) = write(lastOutput or mask)

    /** Desativa no _output_ do UsbPort os bits da `mask`. */
    fun clearBits(mask: Int) = write(lastOutput and mask.inv())

    /** Escreve no _output_ do UsbPort e guarda o que escreveu. */
    private fun write(output: Int){
        UsbPort.write(output)
        lastOutput = output
    }
}

fun main(){
    HAL.init()

    while(true){
        println(HAL.readBits(0xF).toString(2))
        readln()
    }
}
```

## B.2 KBD

### Algoritmo 2: KBD

```
import isel.leic.utils.Time

// Ler teclas. Fun es retornam '0'..'9','A'..'D','#','*' ou NONE
object KBD {

    private const val NONE = 0.toChar()

    private const val KEYPAD = "147*2580369#ABCD"

    private const val KEY_MASK = 0b00001111 // input
    private const val VAL_MASK = 0b00010000 // input

    private const val ACK_MASK = 0b10000000 // output

    /** Inicializa o teclado. */
    fun init(){

        // Garantir que nenhuma tecla acknowledged no nicio
        HAL.clearBits(ACK_MASK)

    }

    /** Retorna de imediato a tecla premida ou `NONE` se n o h tecla premida. */
    fun getKey(): Char {

        // Verificar se existe alguma tecla premida
        if(!HAL.isBit(VAL_MASK)) return NONE

        // Receber tecla premida
        val i = HAL.readBits(KEY_MASK)
        check(i in 0 until KEYPAD.length) { "Invalid_key_code(0x${Integer.toHexString(i)})" }

        // Acknowledge
        HAL.setBits(ACK_MASK)
        while(HAL.isBit(VAL_MASK)) {} // Esperar que acknowledge seja recebido
        HAL.clearBits(ACK_MASK)

        return KEYPAD[i]
    }

    /**
     * Retorna a tecla premida ou `NONE` caso o `timeout` passe.
     * @property timeout Milissegundos at deixar de esperar.
     */
    fun waitKey(timeout: Long): Char {

        val end = Time.getTimeInMillis() + timeout

        while(Time.getTimeInMillis() < end){
            val key = getKey()
            if(key != NONE) return key
        }

        return NONE
    }
}

fun main(){
    KBD.init()
    // println("start")
    // Time.sleep(10000)
    // println("print")
    while(true)
        print(KBD.waitKey(60_000L))
}
```

## B.3 TUI

### Algoritmo 3: KBD

```
import isel.leic.utils.Time
import util.Alignment

object TUI {

  /** 9+ - Para indicar que h mais de 9 cr ditos */
  const val NINE_PLUS = 0.toChar()

  /** Emoticon feliz - Para vit rias */
  const val HAPPY = 1.toChar()

  /** Emoticon triste - Para perdas */
  const val SAD = 2.toChar()

  /** S mbolo de Manuten o */
  const val MAINTENANCE = 3.toChar()

  /** Seta para a direita */
  const val ARROW = 4.toChar()

  /** Inicializa o TUI. */
  fun init(){

    LCD.clear()

    LCD.loadChar(NINE_PLUS, listOf(
      "###_ _",
      "#_ _ _",
      "###_ _",
      "_ _ _ _",
      "###_ _",
      "###_ _",
      "_ _ _ _",
      "_ _ _ _",
      "_ _ _ _"
    ))

    LCD.loadChar(HAPPY, listOf(
      " _ _ _ _",
      " _ _ _ _",
      "_ _ _ _",
      " _ _ _ _",
      "# _ _ _",
      "_ _ _ _",
      " _ _ _ _",
      " _ _ _ _",
      " _ _ _ _"
    ))

    LCD.loadChar(SAD, listOf(
      " _ _ _ _",
      " _ _ _ _",
      "_ _ _ _",
      " _ _ _ _",
      "_ _ _ _",
      "# _ _ _",
      " _ _ _ _",
      " _ _ _ _",
      " _ _ _ _"
    ))

    LCD.loadChar(MAINTENANCE, listOf(
      "#####",
      " _ _ _ _",
      "# _ _ _",
      "#####",
      "# _ _ _",
      "# _ _ _",
      " _ _ _ _",
      " _ _ _ _",
      "#####"
    ))
  }
}
```

```

LCD.loadChar(ARROW, listOf(
    "UUUUU",
    "UU#UU",
    "UUU#U",
    "#####",
    "UUU#U",
    "UU#UU",
    "UUUUU",
    "UUUUU"
))

}

/* ----- LCD ----- */

/**
 * Escreve texto.
 * @property align Alinhamento.
 * @property range Limites do texto.
 */
fun write(text: String, line: Int, align: Alignment = Alignment.CENTER, range: IntRange = 0..LCD.COLS){
    require(line in 0 until LCD.LINES) { "Invalid line." }

    val rangeSize = range.last - range.start
    val startCol = range.start + when(align){
        Alignment.LEFT -> 0
        Alignment.CENTER -> (rangeSize - text.length) / 2
        Alignment.RIGHT -> rangeSize - text.length
    }

    updateLine(line, text, startCol)
}

/**
 * Atualiza a frame de anima o de um texto rotativo.
 *
 * til para mostrar texto maior do que a largura do LCD.
 */
fun scrollText(text: String, line: Int, speed: Int = 250, range: IntRange = 0..LCD.COLS){
    require(line in 0 until LCD.LINES) { "Invalid line." }

    val time = Time.getTimeInMillis() / speed
    val rangeSize = range.last - range.start

    val text = "$text" // Adicionar espaço como margem
    var res = text.substring((time % text.length).toInt()) // Cortar texto
    while(res.length < rangeSize) res += text // Duplicar texto para ocupar espaço inteiro
    res = res.substring(0 until rangeSize) // Manter texto dentro do range

    updateLine(line, res, range.start)
}

fun clear(){
    line0 = List(LCD.COLS){ " " }.joinToString("")
    line1 = List(LCD.COLS){ " " }.joinToString("")
    LCD.clear()
}

private var line0 = List(LCD.COLS){ " " }.joinToString("")
private var line1 = List(LCD.COLS){ " " }.joinToString("")

/** Atualiza as linhas do LCD evitando comandos redundantes. */
private fun updateLine(line: Int, text: String, start: Int = 0){
    // Obter último estado da linha
    val lastLine = when(line){
        0 -> line0
        1 -> line1
        else -> throw IllegalArgumentException("Invalid line.")
    }

    // Sobrescrever linha

```

```

val prefix = if(start > 0) lastLine.substring(0 until start) else ""
val suffix = if(start + text.length <= LCD.COLS) lastLine.substring(start + text.length) else ""
val currLine = (prefix + text + suffix).substring(0 until LCD.COLS)

// Ignorar se n o existir altera es
if(lastLine == currLine) return

// Escrever caract res nos locais necess rios
var col: Int? = null
repeat(LCD.COLS){ i ->
    if(lastLine[i] == currLine[i]) return@repeat

    if(col != i) LCD.cursor(line , i)
    LCD.write(currLine[i])
    col = (col ?: 0) + 1
}

// Atualizar ltime estado da linha
when(line){
    0 -> line0 = currLine
    1 -> line1 = currLine
}

}

/* ----- Keyboard ----- */

/** Retorna de imediato a tecla premida ou `NONE` se n o h tecla premida. */
fun getKey() = KBD.getKey()

}

fun main(){

    HAL.init()
    SerialEmitter.init()
    LCD.init()
    TUI.init()

    while(true){

        TUI.scrollText("If you are seeing this text scrolling , it means that the TUI is working!", 0)

        val key = TUI.getKey()
        if(key != 0.toChar()) TUI.write(key.toString(), 1)

    }

}

```

## B.4 SerialEmitter

### Algoritmo 4: SerialEmitter

```
import isel.leic.utils.Time

/** Envia tramas para os diferentes módulos Serial Receiver */
object SerialEmitter {

  enum class Destination { LCD, ROULETTE }

  // [ NOTA IMPORTANTE! ]
  // 0 bits SEL t m que ser negados
  // Ou seja, para selecionar o LCD, deve-se fazer HAL.clearBits(LCD_SEL)

  private const val LCD_SEL = 0b00000001
  private const val RD_SEL  = 0b00000010

  private const val SDX      = 0b00001000
  private const val SCLK     = 0b00010000

  /** Inicializa o SerialEmitter */
  fun init(){
    HAL.writeBits(
      LCD_SEL or RD_SEL or SDX or SCLK,
      LCD_SEL or RD_SEL // SDX e SCLK devem estar a zero
    )
  }

  /**
   * Envia uma trama para o _Serial Receiver_.
   * @property addr Destino
   * @property data Bits de dados
   * @property size N de bits a enviar
   */
  fun send(addr: Destination, data: Int, size: Int){

    // Ativar destino
    HAL.clearBits(when(addr){
      Destination.LCD -> LCD_SEL
      Destination.ROULETTE -> RD_SEL
    })

    // Enviar dados
    repeat(size){ i ->

      // Preparar SDX
      when((data shr i) and 1){
        1 -> HAL.setBits(SDX)
        0 -> HAL.clearBits(SDX)
      }

      // CLK
      HAL.setBits(SCLK)
      HAL.clearBits(SCLK)

    }

    // Enviar bit de paridade
    when(data.countOneBits() % 2){
      1 -> HAL.clearBits(SDX)
      0 -> HAL.setBits(SDX)
    }

    // CLK
    HAL.setBits(SCLK)
    HAL.clearBits(SCLK)

    // Desativar destino
    HAL.setBits(LCD_SEL or RD_SEL)
  }
}
```

```
fun main(){

    SerialEmitter.init()

    while(true){
        SerialEmitter.send(SerialEmitter.Destination.ROULETTE, readln().toInt(2), 8)
    }

    // println("11100001 - Off")
    // SerialEmitter.send(SerialEmitter.Destination.ROULETTE, 0b11100001, 8)
    // Time.sleep(1000)
    //
    // println("11100000 - On")
    // SerialEmitter.send(SerialEmitter.Destination.ROULETTE, 0b11100000, 8)
    // Time.sleep(1000)
    //
    // // clear
    //
    // println("00011111 - clear digit 0")
    // SerialEmitter.send(SerialEmitter.Destination.ROULETTE, 0b00011111, 8)
    // Time.sleep(1000)
    //
    // println("00111111 - clear digit 1")
    // SerialEmitter.send(SerialEmitter.Destination.ROULETTE, 0b00111111, 8)
    // Time.sleep(1000)
    //
    // println("01011111 - clear digit 2")
    // SerialEmitter.send(SerialEmitter.Destination.ROULETTE, 0b01011111, 8)
    // Time.sleep(1000)
    //
    // println("01111111 - clear digit 3")
    // SerialEmitter.send(SerialEmitter.Destination.ROULETTE, 0b01111111, 8)
    // Time.sleep(1000)
    //
    // println("10011111 - clear digit 4")
    // SerialEmitter.send(SerialEmitter.Destination.ROULETTE, 0b10011111, 8)
    // Time.sleep(1000)
    //
    // println("10111111 - clear digit 5")
    // SerialEmitter.send(SerialEmitter.Destination.ROULETTE, 0b10111111, 8)
    // Time.sleep(1000)
    //
    // // end clear
    //
    // println("00000000 - update digit")
    // SerialEmitter.send(SerialEmitter.Destination.ROULETTE, 0b00000000, 8)
    // Time.sleep(1000)
    //
    // println("11000000 - update")
    // SerialEmitter.send(SerialEmitter.Destination.ROULETTE, 0b11000000, 8)
    // Time.sleep(1000)

    // 0 DXval n o parece ser ativo na simula o
    // Mas isso n o significa que n o est a funcionar
}
```

## B.5 LCD

### Algoritmo 5: LCD

```
import isel.leic.utils.Time

/** Escreve no LCD usando a interface a 4 bits. */
object LCD {

    // Dimens o do display.
    const val LINES = 2
    const val COLS = 16

    /** Define se a interface e S rie ou Paralela */
    private const val SERIAL_INTERFACE = true

    private const val RS_MASK = 0b010000
    private const val E_MASK = 0b100000

    private const val CGRAM_MASK = 0b01000000

    /** Escreve um nibble de comando/dados no LCD em paralelo. */
    private fun writeNibbleParallel(rs: Boolean, data: Int){

        if(rs) HAL.setBits(RS_MASK)
        else HAL.clearBits(RS_MASK)

        HAL.setBits(E_MASK)
        HAL.writeBits(0xF, data)
        HAL.clearBits(E_MASK)
    }

    /** Escreve um nibble de comando/dados no LCD em s rie. */
    private fun writeNibbleSerial(rs: Boolean, data: Int){

        // Adicionar bit de RS
        var data = data shl 1
        if(rs) data = data or 1

        SerialEmitter.send(SerialEmitter.Destination.LCD, data, 5)
    }

    /** Escreve um nibble de comando/dados no LCD. */
    private fun writeNibble(rs: Boolean, data: Int){
        require(data in 0x0..0xF){ "0x${data.toString(16)} is not a valid nibble" }
        return when(SERIAL_INTERFACE){
            true -> writeNibbleSerial(rs, data)
            false -> writeNibbleParallel(rs, data)
        }
    }

    /** Escreve um byte de comando/dados no LCD. */
    private fun writeByte(rs: Boolean, data: Int) {
        require(data in 0x00..0xFF){ "0x${data.toString(16)} is not a valid byte" }
        writeNibble(rs, data shr 4)
        writeNibble(rs, data and 0xF)
    }

    /** Escreve um comando no LCD. */
    private fun writeCMD(data: Int) = writeByte(false, data)

    /** Escreve dados no LCD. */
    private fun writeDATA(data: Int) = writeByte(true, data)

    /** Envia a sequ ncia de inicia o para comunica o a 4 bits. */
    fun init(){

        Time.sleep(20)
        writeNibble(false, 0b0011)
        Time.sleep(5)
        writeNibble(false, 0b0011)
        Time.sleep(1)
        writeNibble(false, 0b0011)
    }
}
```



```

    Time.sleep(1)
    writeNibble(false, 0b0010)

    writeCMD(0b00101000) // function set - DL=0 N=1 F=0
    Time.sleep(1)
    writeCMD(0b00001000) // display off - D=0 C=0 B=0
    Time.sleep(1)
    clear() // clear display
    Time.sleep(5)
    writeCMD(0b00000110) // entry mode set - I/D=1 S=0
    Time.sleep(1)
    writeCMD(0b00001100) // display on - D=1 C=0 B=0
}

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

/** Escreve um carácter na posição corrente. */
fun write(char: Char) = writeDATA(char.code)

/** Envia um comando para posicionar o cursor. */
fun cursor(line: Int, column: Int){
    require(line in 0..LINES - 1) { "Line must be between 0 and ${LINES-1}" }
    require(column in 0..COLS - 1) { "Line must be between 0 and ${COLS-1}" }
    writeCMD(0b10000000 + line * 0x40 + column)
}

/** Envia comando para limpar o ecrã e posicionar o cursor na posição inicial. */
fun clear() = writeCMD(0b00000001)

/**
 * Regista um carácter personalizado.
 * @property char Carácter a associar.
 * @property picture Desenho do carácter 5x8.
 */
fun loadChar(char: Char, picture: List<String>){
    require(char.code in 0b000..0b111){ "Character is more than 3 bits" }
    repeat(8){ i ->
        writeCMD(CGRAM_MASK or (char.code shl 3) or i)
        val code = picture[i].map{ if(it == ' ') '0' else '1' }.joinToString("").toInt(2)
        writeDATA(code)
    }
    writeCMD(0b10)
}
}

fun main(){
    HAL.init()
    SerialEmitter.init()
    LCD.init()

    LCD.loadChar(0.toChar(), listOf(
        "UUUUU",
        "UUUUU",
        "U#U#U",
        "UUUUU",
        "#UUU#",
        "U##U",
        "UUUUU",
        "UUUUU"
    ))

    LCD.write("Hello!_${0.toChar()}")
    LCD.cursor(1, 0)
    LCD.write("0123456789")

    Time.sleep(3000)

    LCD.clear()

```

```
LCD.write("Goodbye...")  
  
Time.sleep(3000)  
  
LCD.clear()  
  
}
```

## B.6 RouletteDisplay

**Algoritmo 6:** RouletteDisplay

```

import isel.leic.utils.Time
import util.Alignment

/** Controla o mostrador de pontua o. */
object RouletteDisplay {

    private const val UPDATE_MASK = 0b00000_110
    private const val ON_MASK = 0b00000_111
    private const val OFF_MASK = 0b00001_111

    // Caracteres especiais
    private const val CHAR_MINUS = 0x10
    private const val CHAR_TOP_RIGHT = 0x11
    private const val CHAR_TOP_LEFT = 0x12
    private const val CHAR_LEFT = 0x13
    private const val CHAR_BOTTOM_LEFT = 0x14
    private const val CHAR_BOTTOM_RIGHT = 0x15
    private const val CHAR_RIGHT = 0x16
    private const val CHAR_LEFT_RIGHT = 0x17
    private const val CHAR_TOP_MID_BOTTOM = 0x18
    private const val CHAR_TOP = 0x19
    private const val CHAR_LEFT_UPPER = 0x1a
    private const val CHAR_LEFT_LOWER = 0x1b
    private const val CHAR_BOTTOM = 0x1c
    private const val CHAR_RIGHT_LOWER = 0x1d
    private const val CHAR_RIGHT_UPPER = 0x1e
    private const val CHAR_EMPTY = 0x1f

    /** Velocidade da anima o */
    private const val ANIM_SPEED = 100

    /** _Frames_ de anima o */
    private val ANIM_FRAMES = listOf(
        listOf(CHAR_LEFT, CHAR_EMPTY, CHAR_EMPTY, CHAR_EMPTY, CHAR_EMPTY,
        CHAR_EMPTY), // [ ]
        listOf(CHAR_TOP_LEFT, CHAR_EMPTY, CHAR_EMPTY, CHAR_EMPTY, CHAR_EMPTY,
        CHAR_EMPTY), // [ ]
        listOf(CHAR_TOP, CHAR_TOP, CHAR_EMPTY, CHAR_EMPTY, CHAR_EMPTY,
        CHAR_EMPTY), // [ -- ]
        listOf(CHAR_EMPTY, CHAR_TOP, CHAR_TOP, CHAR_EMPTY, CHAR_EMPTY,
        CHAR_EMPTY), // [ -- ]
        listOf(CHAR_EMPTY, CHAR_EMPTY, CHAR_TOP, CHAR_TOP, CHAR_EMPTY,
        CHAR_EMPTY), // [ -- ]
        listOf(CHAR_EMPTY, CHAR_EMPTY, CHAR_EMPTY, CHAR_TOP, CHAR_TOP,
        CHAR_EMPTY), // [ -- ]
        listOf(CHAR_EMPTY, CHAR_EMPTY, CHAR_EMPTY, CHAR_EMPTY, CHAR_TOP,
        CHAR_EMPTY), // [ -- ]
        listOf(CHAR_EMPTY, CHAR_EMPTY, CHAR_EMPTY, CHAR_EMPTY, CHAR_EMPTY,
        CHAR_EMPTY), // [ -- ]
        listOf(CHAR_EMPTY, CHAR_EMPTY, CHAR_EMPTY, CHAR_EMPTY, CHAR_EMPTY,
        CHAR_EMPTY), // [ \ ]
        listOf(CHAR_EMPTY, CHAR_EMPTY, CHAR_EMPTY, CHAR_EMPTY, CHAR_EMPTY,
        CHAR_EMPTY), // [ ]
        listOf(CHAR_EMPTY, CHAR_EMPTY, CHAR_EMPTY, CHAR_EMPTY, CHAR_EMPTY,
        CHAR_EMPTY), // [ / ]
        listOf(CHAR_EMPTY, CHAR_EMPTY, CHAR_EMPTY, CHAR_EMPTY, CHAR_EMPTY,
        CHAR_EMPTY), // [ -- ]
        listOf(CHAR_EMPTY, CHAR_EMPTY, CHAR_BOTTOM, CHAR_BOTTOM, CHAR_EMPTY,
        CHAR_EMPTY), // [ -- ]
        listOf(CHAR_EMPTY, CHAR_BOTTOM, CHAR_BOTTOM, CHAR_EMPTY, CHAR_EMPTY,
        CHAR_EMPTY), // [ -- ]
        listOf(CHAR_BOTTOM, CHAR_BOTTOM, CHAR_EMPTY, CHAR_EMPTY, CHAR_EMPTY,
        CHAR_EMPTY), // [ -- ]
        listOf(CHAR_BOTTOM_LEFT, CHAR_EMPTY, CHAR_EMPTY, CHAR_EMPTY, CHAR_EMPTY,
        CHAR_EMPTY) // [ \ ]
    )

    /** Inicializa o mostrador. */
    fun init() {

```

```

        off(true) // Desliga
        off(false) // Liga
        clear() // Limpa
    }

    /** Atualiza a _frame_ de anima o. */
    fun animation(){
        val time = Time.getTimeInMillis() / ANIM_SPEED
        val i = (time % ANIM_FRAMES.size).toInt()
        setAllDigits(ANIM_FRAMES[i])
    }

    /** Escreve um n mero no centro do mostrador. */
    fun setValue(value: Int) = write(value.toString(), Alignment.CENTER)

    /**
     * Envia um comando para desativar/ativar a visualiza o do mostrador.
     *
     * - `true`: Desativar
     * - `false`: Ativar
     */
    fun off(value: Boolean){
        SerialEmitter.send(
            SerialEmitter.Destination.ROULETTE,
            if(value) OFF_MASK else ON_MASK,
            8
        )
    }

    /** Define um d gito numa posi o espec fica. */
    private fun setDigit(pos: Int, value: Int){
        require(pos in 0..5) { "pos_must_be_between_0_to_5" }
        require(value in 0x00..0x1f) { "value_must_be_from_0x00_to_0x1f" }

        // Evita atualizar o d gito caso seja igual ao ltimo estado
        if(lastState[pos] == value) return

        // Atualizar pr ximo estado
        nextState[pos] = value

        // Enviar para o mostrar
        SerialEmitter.send(SerialEmitter.Destination.ROULETTE, (value shl 3) or (5 - pos), 8)
    }

    /** Define todos os d gitos a partir de uma lista e atualiza */
    private fun setAllDigits(digits: List<Int>){
        repeat(6){ i -> setDigit(i, digits[i]) }
        update()
    }

    /** Define todos os d gitos e atualiza. */
    private fun setAllDigits(d0: Int, d1: Int, d2: Int, d3: Int, d4: Int, d5: Int) =
        setAllDigits(listOf(d0, d1, d2, d3, d4, d5))

    /** Escreve caracteres no mostrador com alinhamento. */
    fun write(string: String, align: Alignment = Alignment.LEFT){
        require(Regex("[_0-9a-fA-F]{0,6}\\$").matches(string)) { "string_must_be_up_to_6_characters_long_and" }

        // Calcular nicio dependendo do alinhamento
        val startPos = when(align){
            Alignment.LEFT -> 0
            Alignment.CENTER -> (6 - string.length) / 2
            Alignment.RIGHT -> 6 - string.length
        }

        // Limpar tudo
        repeat(6){ i -> setDigit(i, CHAR_EMPTY) }

        // Escrever texto
        repeat(string.length){ i ->
            setDigit(startPos + i, when(string[i]){
                '-' -> CHAR_MINUS
                '_' -> CHAR_BOTTOM
            })
        }
    }

```

```

        '0' -> 0
        '1' -> 1
        '2' -> 2
        '3' -> 3
        '4' -> 4
        '5' -> 5
        '6' -> 6
        '7' -> 7
        '8' -> 8
        '9' -> 9
        'a', 'A' -> 0xa
        'b', 'B' -> 0xb
        'c', 'C' -> 0xc
        'd', 'D' -> 0xd
        'e', 'E' -> 0xe
        'f', 'F' -> 0xf
        ' ' -> CHAR_EMPTY
        else -> throw IllegalStateException("Unexpected_${string[i]}_character!")
    })
}

update()

}

private var lastState = MutableList(6) { 0 }
private var nextState = MutableList(6) { 0 }

/**
 * Atualiza mostrador, caso haja alterações.
 * @property forced Atualiza mesmo não havendo alterações
 */
private fun update(forced: Boolean = false){

    // Evitar atualizar caso não seja forçado e não exista mudanças
    if(!forced && lastState == nextState) return

    // Enviar comando para atualizar
    SerialEmitter.send(SerialEmitter.Destination.ROULETTE, UPDATE_MASK, 8)

    // Atualizar o último estado
    repeat(6){ i-> lastState[i] = nextState[i] }

}

/** Limpa o mostrador. */
fun clear() = write("uuuuuu")
}

fun main(){

    RouletteDisplay.init()

    RouletteDisplay.write("2-5", Alignment.CENTER)
    Time.sleep(1000)

    val end = Time.currentTimeMillis() + 5000
    while(Time.currentTimeMillis() < end)
        RouletteDisplay.animation()

    RouletteDisplay.setValue(-3)

}

```

## B.7 RouletteGame

### Algoritmo 7: RouletteGame

```
import isel.leic.utils.Time
import util.Alignment
import util.RouletteBoard
import kotlin.math.ceil

object RouletteGame {

    /** N de moedas a gastar. */
    private var credits = 0

    /** Deteta novas moedas e adiciona ao jogo. */
    private fun checkNewCoins() {

        // Detetar nova moeda
        val coin = CoinAcceptor.get()
        if(coin == 0) return

        // Adicionar aos cr ditos
        credits += coin
    }

    fun init(){

        // Inicializa o

        HAL.init()
        SerialEmitter.init()
        LCD.init()
        RouletteDisplay.init()
        KBD.init()
        TUI.init()

        CoinDeposit.init()
        Statistics.init()

        // Ecr Inicial

        while(true){

            checkNewCoins()

            // --- LCD ---

            // T tulo
            TUI.write("RouletteGame", 0)

            // Cr ditos
            val creditsStr = "₹${credits}$"
            TUI.write(creditsStr, 1, Alignment.RIGHT)

            // Instru es
            TUI.scrollText(
                if(credits > 0) "Click*to play!" else "Insert coins!",
                1,
                range = 0..(LCD.COLS - creditsStr.length)
            )

            // --- KB ---

            // Come ar o jogo
            if(TUI.getKey() == '*' && credits > 0) startGame()

            // Manuten o
            if(M.check()) M.maintenanceMode()
        }
    }
}
```

```

}

fun startGame(maintenance: Boolean = false){

    // Apostas

    val bets = RouletteBoard.default(0)
    var lastBet = Time.getTimeInMillis()
    var betEnd: Long? = null

    TUI.clear()

    while(true){

        checkNewCoins()

        // --- LCD ---

        val betSum = bets.toList().sum()

        if(betSum == 14 * 9) {

            // 0 jogador n o pode apostar mais
            // Mostrar instru o para come ar a jogar
            var msg = "Max_bets!"
            if(betEnd == null) msg += "Click#to_start."
            TUI.scrollText(msg, 0, range = 0..LCD.COLS - 2)

        } else if(Time.getTimeInMillis() - lastBet > 5e3 && credits == 0){

            // 0 jogador est inativo e necessita de mais cr ditos
            val msg = if(betEnd == null) "Click#to_start_or_insert_coins!" else "Insert_coins_to_keep_be"
            TUI.scrollText(msg, 0, range = 0..LCD.COLS - 2)

        } else if(Time.getTimeInMillis() - lastBet > 5e3 && betSum > 0) {

            // 0 jogador est inativo
            // Mostrar instru o para come ar a jogar
            val msg = if(betEnd == null) "Click#to_start_or_keep_betting!" else "You_can_still_bet!"
            TUI.scrollText(msg, 0, range = 0..LCD.COLS - 2)

        } else if(betSum > 0){

            // 0 jogador j apostou
            // Mostrar apostas do utilizador
            TUI.write(
                bets.toList().joinToString("") { if (it == 0) " " else it.toString() },
                0, Alignment.LEFT
            )

        } else{

            // 0 jogador ainda n o apostou
            // Mostrar instru o para apostar
            TUI.scrollText("Click_on_these_keys_to_bet!", 0, range = 0..LCD.COLS - 2)

        }

        val cornerIcon = when {
            betEnd != null -> ceil((betEnd - Time.getTimeInMillis()) / 1e3).toInt().toString()
            !maintenance -> "$"
            else -> null
        }
        if(cornerIcon != null) TUI.write(cornerIcon, 0, Alignment.RIGHT)

        TUI.write("0123456789ABCD_${when{
            maintenance->TUI.MAINTENANCE
            credits->9->TUI.NINE_PLUS
            else->credits
        }}", 1)

        // --- KB ---

        val key = TUI.getKey()
    
```

```
// Apostas
if(
    key in RouletteBoard.SLOTS &&
    (credits > 0 || maintenance) && // Tem créditos ou está em modo manutençã
    bets[key] < 9 // Não chegou ao limite de apostas dessa tecla
){

    bets[key]++
    if(!maintenance){

        // Retirar dos créditos
        credits--

        // Adicionar ao depósito
        CoinDeposit.addCoin()

    }

    lastBet = Time.getTimeInMillis()

}

// Começar o jogo
if(
    key == '#' &&
    bets.toList().sum() > 0 && // Existe alguma aposta
    betEnd == null // Jogo ainda não começou
) betEnd = Time.getTimeInMillis() + 5000L // Acabar as apostas em 5 segundos

// O jogo já começou
if(betEnd != null){
    RouletteDisplay.animation()
    if(Time.getTimeInMillis() > betEnd) break // Acabar as apostas
}

}

// Animação
TUI.clear()

val animEnd = Time.getTimeInMillis() + (1000L..5000L).random()
while(Time.getTimeInMillis() < animEnd){

    RouletteDisplay.animation()

    TUI.write("Good luck!", 0)
    TUI.scrollText("Not accepting more bets.", 1)

}

// Fim
TUI.clear()

val winSlot = (0x0..0xD).random()
val winCredits = bets[winSlot] * 2

RouletteDisplay.write("${winSlot.toString(16)}░░░${winCredits.toString().padStart(2, '░')}")

val winEnd = Time.getTimeInMillis() + 5000L
while(Time.getTimeInMillis() < winEnd){
    TUI.write(if(bets[winSlot] > 0) "Congrats! ${TUI.HAPPY}" else "Oh no! ${TUI.SAD}", 0)
    TUI.scrollText(if(bets[winSlot] > 0) "You won ${winCredits} credits!" else "Better luck next time!")
}

if(!maintenance){

    credits += winCredits

    CoinDeposit.cashOut(winCredits)

    // Atualizar info e estatísticas
```



```
        Statistics.addResult(winSlot, winCredits)
        CoinDeposit.addGamePlayed()

    }

    RouletteDisplay.clear()

}

fun main() = RouletteGame.init()
```

## B.8 CoinAcceptor

### Algoritmo 8: CoinAcceptor

```
object CoinAcceptor {  
  
    private const val COIN_MASK = 0b01000000  
    private const val COIN_ID_MASK = 0b00100000  
    private const val COIN_ACCEPT_MASK = 0b01000000  
  
    /**  
     * Deteta novas moedas e retorna o seu valor.  
     *  
     * Retorna `0`, caso n o tiver nenhuma moeda.  
     */  
    fun get(): Int {  
  
        // Verificar se existe uma moeda  
        if (!HAL.isBit(COIN_MASK)) return 0  
  
        // Verificar o valor da moeda  
        val coin = if (HAL.isBit(COIN_ID_MASK)) 4 else 2  
  
        // Aceitar moeda  
        HAL.setBits(COIN_ACCEPT_MASK)  
        while (HAL.isBit(COIN_MASK)) {}  
        HAL.clearBits(COIN_ACCEPT_MASK)  
  
        return coin  
    }  
}  
  
fun main() {  
    while (true) {  
        val coins = CoinAcceptor.get()  
        if (coins != 0) println("${coins}$")  
    }  
}
```

## B.9 CoinDeposit

### Algoritmo 9: CoinDeposit

```
import util.GRAY
import util.RED
import util.RESET
import java.io.File
import kotlin.math.max
import kotlin.math.min

data class Info(
    /** N de jogos realizados. */
    var gameCount: Int,
    /** N de moedas guardadas no cofre do moedeiro. */
    var coinCount: Int
)

object CoinDeposit {
    private const val INFO_PATH = "statistics.txt"
    private var _info: Info? = null
    val info: Info get() = _info!!

    /**
     * Carrega o `Info` que esteja no seu ficheiro.
     *
     * Caso exista ocorra algum erro, retornado um `Info` inicial.
     */
    fun init(){
        try{
            val data = FileAccess.read(INFO_PATH)
            assert(data.size == 2) { "Unexpected amount of data!" }

            val (gameCount, coinCount) = data.map { it.toIntOrNull() }
            assert(gameCount != null && coinCount != null){ "Unexpected type of data!" }

            _info = Info(gameCount!!, coinCount!!)
        } catch (e: Exception){
            println("${RED}LoadInfoError:${RESET} ${e.message} ${GRAY}- Returning default info...${RESET}")
            resetInfo()
        }
    }

    /** Incrementa o n de jogos realizados. */
    fun addGamePlayed() = _info!!.gameCount++

    /** Incrementa o n de moedas guardadas no cofre do moedeiro */
    fun addCoin() = _info!!.coinCount++

    /** Subtrai o n de moedas guardadas no cofre do moedeiro */
    fun cashOut(coins: Int){
        _info!!.coinCount -= coins
    }

    /** Reinicia `Info`. */
    fun resetInfo(){
        _info = Info(0, 0)
    }

    /** Guarda `Info` no seu ficheiro. */
    fun saveInfo(){
        try{
            FileAccess.write(INFO_PATH, "${info.gameCount}\n${info.coinCount}")
        } catch (e: Exception){
            println("${RED}SaveInfoError:${RESET} ${e.message}")
        }
    }
}
```

## B.10 Maintenance

### Algoritmo 10: Maintenance

```
import isel.leic.utils.Time
import util.Alignment
import util.RouletteBoard
import kotlin.system.exitProcess

object M {

    private const val M_MASK = 0b10000000

    /** Retorna o estado de manuten o */
    fun check() = HAL.isBit(M_MASK)

    // Modo de Manuten o
    fun maintenanceMode(){

        TUI.clear()

        while(true){

            TUI.write("${TUI.MAINTENANCE}AINTENANCE",0)

            // Lista de Comandos
            TUI.scrollText("*_${TUI.ARROW}_Test_|" +
                "_A_${TUI.ARROW}_Info_|" +
                "_C_${TUI.ARROW}_Results_|" +
                "_D_${TUI.ARROW}_Off_|" +
                "_#_${TUI.ARROW}_Back_|",1, 200)

            val key = TUI.getKey()

            // Keys dos Comandos
            when (key) {
                '*' -> RouletteGame.startGame(true)
                'A' -> maintenanceInfo()
                'C' -> maintenanceResults()
                'D' -> maintenanceOff()
            }

            // Sair do modo de manuten o
            if(!check()) return

        }

    }

    fun maintenanceInfo(){

        TUI.clear()

        while(true){

            TUI.write("Coins:_${CoinDeposit.info.coinCount}",0, Alignment.LEFT)
            TUI.write("Games:_${CoinDeposit.info.gameCount}",1, Alignment.LEFT)

            val key = TUI.getKey()

            if(key == '*'){
                CoinDeposit.resetInfo()
                TUI.clear()
            }

            // Sair do comando
            if(key == '#'){
                TUI.clear()
                return
            }

        }

    }

}
```

```

}

fun maintenanceResults(){
    TUI.clear()

    var selectedSlot: Char? = null

    while(true){
        if(selectedSlot == null){
            TUI.write("Click on a slot", 0)
            TUI.write("to see results", 1)
        } else {
            TUI.write("Wins: ␣${Statistics.results[selectedSlot].winCount}", 0, Alignment.LEFT)
            TUI.write("Coins: ␣${Statistics.results[selectedSlot].coinCount}", 1, Alignment.LEFT)
            TUI.write(selectedSlot.toString(), 0, Alignment.RIGHT)
        }

        val key = TUI.getKey()

        if(key in RouletteBoard.SLOTS){
            selectedSlot = key
            TUI.clear()
        }

        if (key == '*'){
            Statistics.resetResults()

            TUI.clear()
            TUI.write("Results reseted!", 0)
            Time.sleep(2000L)
            TUI.clear()
        }

        // Sair do comando
        if(key == '#'){
            TUI.clear()
            return
        }
    }
}

fun maintenanceOff(){
    TUI.clear()

    while (true){
        TUI.write("Shut off?",0)
        TUI.write("␣${TUI.ARROW}␣Yes␣|␣#␣${TUI.ARROW}␣No",1)

        val key = TUI.getKey()

        // Guardar coins e resultados e depois desligar
        if(key == '*'){
            TUI.clear()
            TUI.write("Shutting off...", 0)

            CoinDeposit.saveInfo()
            Statistics.saveResults(Statistics.results)

            Time.sleep(1000)
            TUI.clear()

            exitProcess(0)
        }

        // Sair do comando
        if(key == '#'){

```

```
        TUI.clear()
        return
    }
}
}
}

fun main(){
    var lastState: Boolean? = null

    while(true){
        val state = M.check()
        if(state == lastState) continue
        lastState = state

        println("Maintenance is ${if(state) "ON" else "OFF"}")
    }
}
```

## B.11 FileAccess

### Algoritmo 11: FileAccess

```
import util.GRAY
import util.RED
import util.RESET
import util.RouletteBoard
import util.toRouletteBoard
import java.io.File

object FileAccess {

    fun read(path: String) = File(path).readLines()

    fun write(path: String, content: String) = File(path).writeText(content)

}
```

## B.12 RouletteBoard

### Algoritmo 12: RouletteBoard

```
package util

/** til para organizar dados associados a cada opção de aposta. */
data class RouletteBoard<E>() {
    var slot0: E,
    var slot1: E,
    var slot2: E,
    var slot3: E,
    var slot4: E,
    var slot5: E,
    var slot6: E,
    var slot7: E,
    var slot8: E,
    var slot9: E,
    var slotA: E,
    var slotB: E,
    var slotC: E,
    var slotD: E
}: Iterable<E> {

    companion object Factory {

        const val SLOTS = "0123456789ABCD"

        /** Cria um RouletteBoard que tenta todos os dados iguais. */
        fun <E> default(value: E) = List(14){ value }.toRouletteBoard()

    }

    /** Acesso via n meros hexadecimais. */
    operator fun get(i: Int) = when(i){
        0x0 -> slot0
        0x1 -> slot1
        0x2 -> slot2
        0x3 -> slot3
        0x4 -> slot4
        0x5 -> slot5
        0x6 -> slot6
        0x7 -> slot7
        0x8 -> slot8
        0x9 -> slot9
        0xA -> slotA
        0xB -> slotB
        0xC -> slotC
        0xD -> slotD
        else -> throw IndexOutOfBoundsException("$i is outside de 0x0..0xD range.")
    }

    /** Acesso via caracteres. */
    operator fun get(i: Char) = when(i){
        '0' -> slot0
        '1' -> slot1
        '2' -> slot2
        '3' -> slot3
        '4' -> slot4
        '5' -> slot5
        '6' -> slot6
        '7' -> slot7
        '8' -> slot8
        '9' -> slot9
        'A' -> slotA
        'B' -> slotB
        'C' -> slotC
        'D' -> slotD
        else -> throw IndexOutOfBoundsException("$i is outside de '0'..'D' range.")
    }

    /** Atribui o via n meros hexadecimais. */
    operator fun set(i: Int, value: E) = when(i){
```



```

0x0 -> slot0 = value
0x1 -> slot1 = value
0x2 -> slot2 = value
0x3 -> slot3 = value
0x4 -> slot4 = value
0x5 -> slot5 = value
0x6 -> slot6 = value
0x7 -> slot7 = value
0x8 -> slot8 = value
0x9 -> slot9 = value
0xA -> slotA = value
0xB -> slotB = value
0xC -> slotC = value
0xD -> slotD = value
else -> throw IndexOutOfBoundsException("$i_is_outside_de_0x0..0xD_range.")
}

/** Atribui o via caracteres. */
operator fun set(i: Char, value: E) = when(i){
    '0' -> slot0 = value
    '1' -> slot1 = value
    '2' -> slot2 = value
    '3' -> slot3 = value
    '4' -> slot4 = value
    '5' -> slot5 = value
    '6' -> slot6 = value
    '7' -> slot7 = value
    '8' -> slot8 = value
    '9' -> slot9 = value
    'A' -> slotA = value
    'B' -> slotB = value
    'C' -> slotC = value
    'D' -> slotD = value
    else -> throw IndexOutOfBoundsException("$i_is_outside_de_'0'..'D'_range.")
}

/** Convers o para lista de dados. */
fun toList() = listOf(
    slot0,
    slot1,
    slot2,
    slot3,
    slot4,
    slot5,
    slot6,
    slot7,
    slot8,
    slot9,
    slotA,
    slotB,
    slotC,
    slotD
)

/** Iterador */
override fun iterator() = toList().iterator()
}

/** Convers o de lista para RouletteBoard. */
fun <E> List<E>.toRouletteBoard(): RouletteBoard<E> {
    require(size == 14) { "List_must_have_exactly_14_elements_(0x0..0xD)" }
    return RouletteBoard(
        this[0x0],
        this[0x1],
        this[0x2],
        this[0x3],
        this[0x4],
        this[0x5],
        this[0x6],
        this[0x7],
        this[0x8],
        this[0x9],
        this[0xA],
        this[0xB],
        this[0xC],
        this[0xD]
    )
}

```

```
    this[0xB] ,  
    this[0xC] ,  
    this[0xD]  
    )  
}
```