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

Licenciatura em Engenharia Informática e de Computadores

Jogo da Roleta

(Roulette Game)

A46378 Berto Barata
A44787 Gonçalo Garcia
A44776 João Gomes

Professores:
Pedro Miguens Matutino (pedro.miguens@isel.pt)
Nuno Sebastião (nuno.sebastiao@isel.pt)

Projeto de
Laboratório de Informática e Computadores
2020 / 2021 inverno

22 de outubro de 2020

Índice

Introdução

Neste projeto implementa-se o jogo da Roleta (*Roulette Game*), no qual a roleta compreende números entre 0 e 9, um jogador realiza apostas premindo as teclas de um teclado correspondentes aos números em que pretende apostar. Por cada aposta é debitado um crédito ao saldo acumulado do jogador, podendo o jogador apostar mais do que um crédito num mesmo número. Os créditos são obtidos pela introdução de moedas no moedeiro, este só aceita moedas de 1.00€, que corresponde a dois créditos. O sistema que implementa o jogo será constituído por: um PC (*Control*); um teclado de 12 teclas; um moedeiro (*Coin Acceptor*); um mostrador *Liquid Cristal Display* (*LCD*) de duas linhas com 16 caracteres; um mostrador da roleta (*Roulette Display*) e uma chave de manutenção designada por *M*, para colocação do sistema em modo de Manutenção. Na Figura 1 apresenta-se o diagrama de blocos do jogo da Roleta.

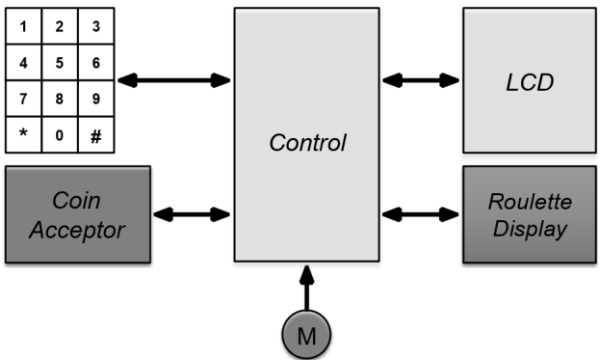

Figura 1 – Diagrama de blocos do jogo da Roleta (*Roulette Game*)

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

Jogo – O jogo inicia-se quando é premida a tecla '*' e existem créditos disponíveis. Utilizando as teclas numéricas (09) realizam-se as apostas, retirando-se um crédito ao saldo do jogador por cada aposta realizada. O jogador termina as apostas premindo a tecla '#', o que dá início ao sorteio. Durante um tempo aleatório, o sistema simula o girar da Roleta no *Roulette Display*, permitindo ainda realizar apostas até 5 segundos antes desta parar. Ao parar a Roleta, o número sorteado é apresentado no *Roulette Display* e os créditos obtidos na jogada são apresentados no LCD. Os créditos obtidos são acumulados após 5 segundos ao saldo do jogador, também apresentado no LCD.

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

- **Teste** – Premindo a tecla '*' inicia-se um jogo sem créditos e sem contabilizar os números sorteados.
- **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.
- **Consultar a lista de números sorteados** – Carregando na tecla '0' permite-se a listagem dos números sorteados.

- **Iniciar a lista de números sorteados –** Premindo a tecla ‘0’ e em seguida a tecla ‘*’, o sistema inicia um novo ciclo de estatística de números sorteados.

- **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 Números Sorteados, 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 Números Sorteados, que contém o número de saídas e os prémios atribuídos por cada número. Os dois ficheiros devem ser carregados para o sistema no seu processo de arranque.

Arquitetura do sistema

O sistema é implementado numa solução híbrida de *hardware* e *software*, como apresentado no diagrama de blocos da Figura 2. A arquitetura proposta é constituída por quatro módulos principais: *i*) um leitor de teclado, designado por *Keyboard Reader*; *ii*) um módulo de interface com o *LCD* e com o mostrador da roleta, designado por *Serial Output Controller* (*SOC*); *iii*) um moedeiro, designado por *Coin Acceptor*; e *iv*) um módulo de controlo, designado por *Control*. Os módulos *i*) e *ii*) são implementados em *hardware*, o moedeiro será simulado utilizando um interruptor e um LED, enquanto o módulo de controlo é implementado em *software* escrito usando linguagem Java e executado num PC.

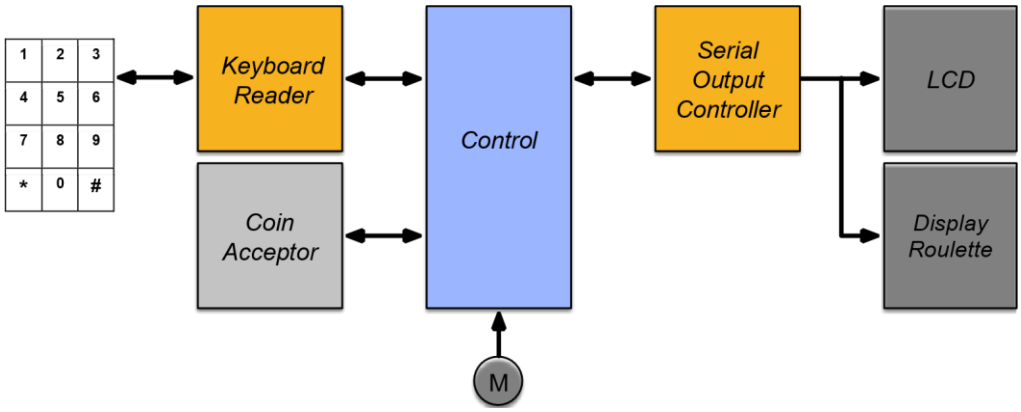

Figura 2 – Arquitetura do sistema que implementa o jogo da Roleta (*Roulette Game*)

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 internamente, até ao limite de dois códigos. O módulo *Control* processa os dados e envia a informação a apresentar no *LCD* através do módulo *SOC*. O *Roulette Display* é atuado pelo módulo *Control*, através do módulo *SOC*. Por razões de ordem física, e por forma a minimizar o número de fios de interligação, a comunicação entre o módulo *Control* e o módulo *SOC* é realizada através de um protocolo série.
A implementação do módulo *Control* foi realizada em *software*, usando a linguagem Java e seguindo a arquitetura lógica apresentada na Figura 3.

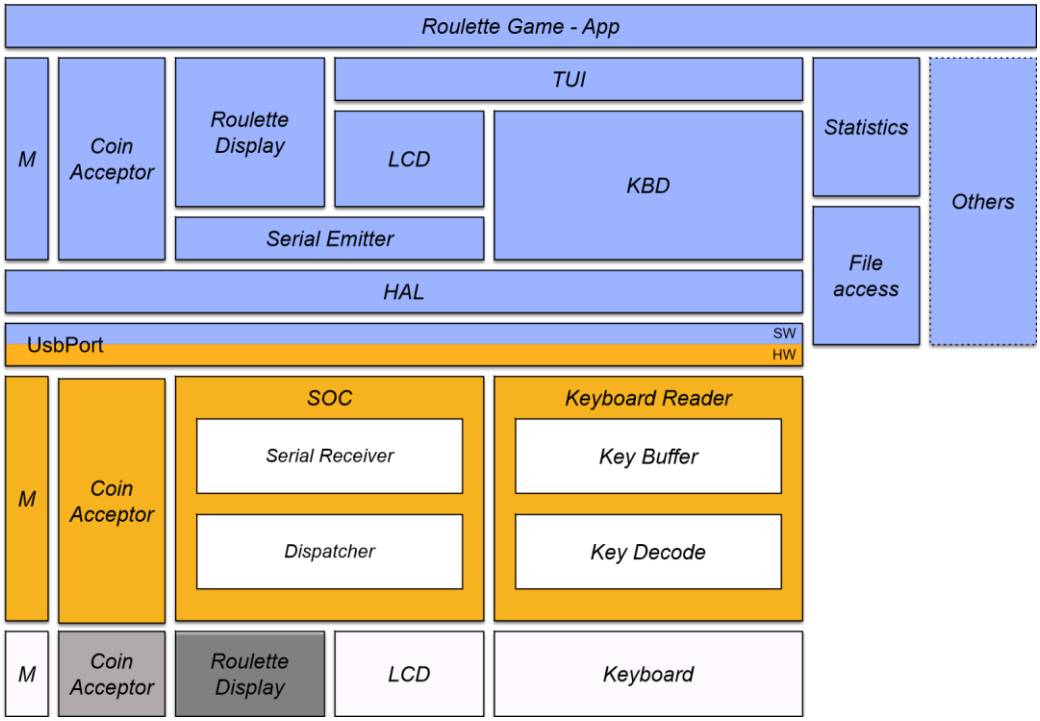

Figura 3 – Diagrama lógico do Jogo da Roleta (*Roulette Game*)

O módulo Coin Acceptor simula um moedeiro, sinalizando ao módulo Control que foi inserida uma moeda através da ativação do sinal Coin. A entidade consumidora informa o CoinAcceptor que já contabilizou a moeda ativando o sinal accept. Neste módulo do trabalho foi realizada a classe CoinAcceptor para servir de interface virtual para a sinalização das moedas inserida; em termos de hardware foi apenas adicionado um interruptor para gerar o sinal Coin, e o sinal de confirmação (accept) que será visualizado num LED.

Interligações entre o *HW* e *SW*

A interligação entre o software e o hardware é conseguida através da utilização do USB_PORT e da ATB. Para enviar dados tanto para o display de 7-segmentos como o LCD são utilizadas as funções input e output. São também utilizadas três PAL's ATF750C descritas com linguagem hardware (CUPL) e três breadboards para realizar as funções pretendidas. A necessidade do uso da ATB deve-se ao facto de ser preciso um clock e interruptores para desempenhas as funções tanto do M como do CoinAcceptor. Desta forma pudemos fazer tanto o hardware e o software interagir entre si de modo a realizar corretamente as funções espectáveis pelo jogo Roulette.

Código Java da classe *HAL*

```
import isel.leic.UsbPort;

// Hardware Abstraction Layer.
public class HAL {     private static int outputPort; // The value
on the output port.
    public static void main(String[] args) {
init();
    }
    public static void init() {
outputPort = 0;          out(outputPort);
    }
    // Checks, on the input port, if a specific bit is set.
public static boolean isBit(int mask) {          return 0 !=
readBits(mask);
    }
    // Reads, from the input port, the bits specified by a mask.
public static int readBits(int mask) {          return in() &
mask;
    }
    // Writes, to the output port, on the bits specified by a mask (while keeping
the others), a value.
    public static void writeBits(int mask, int value) {
outputPort &= ~mask;          outputPort |= value &
mask;          setBits(outputPort);
clrBits(~outputPort);
    }
    // Sets, on the output port, the bits specified by a mask (and keeps the
others).
    public static void setBits(int mask) {
outputPort |= mask;          out(outputPort);
    }
    // Resets, on the output port, the bits specified by a mask (and keeps the
others).
    public static void clrBits(int mask) {
outputPort &= ~mask;          out(outputPort);
    }
    private static int in() {          return ~UsbPort.in(); //
UsbPort.in() reads from the input port.
    }
    private static void out(int value) {
        UsbPort.out(~value); // UsbPort.out() writes to the output port.
    }
}
```

Código Java da classe *KBD*

```
import isel.leic.utils.Time;
// Keyboard.
public class KBD {
    public static final char NONE = 0;
private static final char[] KEYBOARD = {
            '1', '4', '7',
            '*', '2', '5',
            '8', '0', '3',
            '6', '9', '#'
    };
    private static final int OUT_REGISTER_MASK_IN_PORT = 0x0F;
private static final int DVAL_MASK_IN_PORT = 0x10; // Data Valid.
private static final int ACK_MASK_OUT_PORT = 0x80; // Acknowledge.
    public static void main(String[] args) {
        HAL.init();
init();
    }
    public static void init() {
        HAL.clrBits(ACK_MASK_OUT_PORT);
    }
    // Gets the key being pressed. Returns NONE if no key is being pressed.
public static char getKey() {         char key =
HAL.isBit(DVAL_MASK_IN_PORT) ?
KEYBOARD[HAL.readBits(OUT_REGISTER_MASK_IN_PORT)] : NONE;         if (NONE
== key) {             return NONE;
        }
        HAL.setBits(ACK_MASK_OUT_PORT);
while (HAL.isBit(DVAL_MASK_IN_PORT));
HAL.clrBits(ACK_MASK_OUT_PORT);
return key;
    }
    // Waits for timeout milliseconds for a key press. Returns the pressed key or
NONE if no key was pressed during the specified
    // time span.     public static char
waitKey(long timeout) {         timeout +=
Time.getTimeInMillis();         char key;
        while (Time.getTimeInMillis() <= timeout) {
if (NONE != (key = getKey())) {
return key;
            }
}
        return NONE;
    }
}
```

Código Java da classe *LCD*

```
import isel.leic.utils.Time;
// Liquid Cristal Display.
public class LCD {
    public static final int LINES = 2, COLS = 16;
private static final boolean SERIAL_INTERFACE = true;
private static final int DATA_MASK_OUT_PORT = 0x0F;
private static final int RS_MASK_OUT_PORT = 0x10; // Register Select.
private static final int E_MASK_OUT_PORT = 0x20; // Enable.
    public static void main(String[] args) {
        HAL.init();
        SerialEmitter.init();
init();
    }
    private static void writeNibbleParallel(boolean rs, int data) {
if (rs) {
            HAL.setBits(RS_MASK_OUT_PORT);
} else {
            HAL.clrBits(RS_MASK_OUT_PORT);
        }
        HAL.setBits(E_MASK_OUT_PORT);
        HAL.writeBits(DATA_MASK_OUT_PORT, data);
        HAL.clrBits(E_MASK_OUT_PORT);
    }
    private static void writeNibbleSerial(boolean rs, int data) {
        SerialEmitter.send(SerialEmitter.Destination.LCD, data << 1 | (rs ? 1 : 0));
    }
    private static void writeNibble(boolean rs, int data) {
if (SERIAL_INTERFACE) {
            writeNibbleSerial(rs, data);
        } else {
            writeNibbleParallel(rs, data);
        }
}
    private static void writeByte(boolean rs, int data) {
writeNibble(rs, data >> 4);         writeNibble(rs,
data);
    }
    private static void writeCMD(int data) {
writeByte(false, data);
    }
    private static void writeDATA(int data) {
writeByte(true, data);
    }
```

```
public static void init() {
        writeNibble(false, 3);
        Time.sleep(5); // // 4,1 ms until next write.
        writeNibble(false, 3);
        Time.sleep(1); // 100 us until next write.
        writeNibble(false, 3);
        writeNibble(false, 2);
        writeCMD(0x2C);
        writeCMD(0x08);
        writeCMD(0x01);
        writeCMD(0x06);
        writeCMD(0x0F);
        writeCMD(0x0C); // Hide the cursor.
    }
    public static void write(char c) {
writeDATA(c);
    }
    public static void write(String txt) {
for (int i = 0; i < txt.length(); i++) {
writeDATA(txt.charAt(i));
        }
    }
    // Cells: 0-7.     public static void setCustomChar(int
cell, int c[]) {         writeCMD(0x40 + (cell * 8)); // Set
the CGRAM address.
        for (int i : c) {
writeDATA(i);
        }
    }
    // Lines and columns start at 1.
    public static void cursor(int line, int col) {
        writeCMD(0x80 | (1 == line ? col - 1 : col - 1 + 0x40));
    }
    public static void clear() {
writeCMD(0x01);
    }
}
```

Código Java da classe *TUI*

```
// Text User Interface.
public class TUI {
public enum Alignment {
        CENTER,
        LEFT,
        RIGHT
}
    public static final char NONE = KBD.NONE;
public static void main(String[] args) {
        HAL.init();
        KBD.init();
LCD.init();         init();
    }
    public static void init() {
clearLCD();
    }
    public static void clearLCD() {
        LCD.clear();
    }
    // Lines start at 1.
    public static void printAlignedStringLCD(int line, Alignment alignment, String
txt) {
        if (Alignment.CENTER == alignment) {
            LCD.cursor(line, (LCD.COLS - txt.length()) / 2 + 1);
        } else if (Alignment.LEFT == alignment) {
            LCD.cursor(line, 1);
        } else if (Alignment.RIGHT == alignment) {
            LCD.cursor(line, LCD.COLS - txt.length() + 1);
        }
        LCD.write(txt);
    }
    // Lines and columns start at 1.
    public static void printCharLCD(int line, int col, char c) {
        LCD.cursor(line, col);
        LCD.write(c);
    }
    // Cells: 0-7.     public static void
setCustomCharLCD(int cell, int c[]) {
        LCD.setCustomChar(cell, c);
    }
    public static char waitKeyKBD(long timeout) {
return KBD.waitKey(timeout);
    }
    public static char getKeyKBD() {
return KBD.getKey();
    }
}
```

Código Java da classe *FILEACCESS*

```
import java.io.*;
import java.util.ArrayList;

public class FileAccess {
    public static void main(String[] args) {
    }
    public static ArrayList<String> read(String fileName) {
ArrayList<String> content = null;
        try (BufferedReader bR = new BufferedReader(new FileReader(fileName))) {
content = getLines(bR);          } catch (IOException e) {
            e.printStackTrace();
}
        return content;
    }
    private static ArrayList<String> getLines(BufferedReader bR) {
        ArrayList<String> content = null;
try {
            String line;
            content = new ArrayList<>();
while (null != (line = bR.readLine())) {
content.add(line);
            }
        } catch (IOException e) {
            e.printStackTrace();
}
        return content;
    }
    public static void write(String fileName, ArrayList<String> content) {
try (BufferedWriter bW = new BufferedWriter(new FileWriter(fileName))) {
setLines(content, bW);          } catch (IOException e) {
            e.printStackTrace();
        }
}
    private static void setLines(ArrayList<String> content, BufferedWriter bW) {
try {
            for (String s : content) {
bW.write(s);                  bW.newLine();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Código Java da classe *STATISTICS*

```
public class Statistic {     private int number; // The number, 0-
9.     private int count; // How many times the number was drawn.
private int amount; // The amount won by betting on the number.
    public Statistic(int number) {
this.number = number;
    }
    public Statistic(Statistic statistic) {
number = statistic.getNumber();
count = statistic.getCount();          amount
= statistic.getAmount();
    }
    public int getNumber() {
return number;
    }
    public int getCount() {
return count;
    }
    public int getAmount() {
return amount;
    }
    public void setCount(int count) {
this.count = count;
    }
    public void incrementCount(int inc) {
count += inc;
    }
    public void setAmount(int amount) {
this.amount = amount;
    }
    public void incrementAmount(int inc) {
amount += inc;
    }
}
```

Código Java da classe *Roulette Display*

```
import isel.leic.utils.Time;

public class RouletteDisplay {
    private static final int CLEAR_DISPLAY = 0x17;
    private static final char ANIMATION[] = {0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F};
private static final int ANIMATION_MAX_INDEX = ANIMATION.length - 1;     private
static final long ANIMATION_TIMEOUT_MS = 300;     private static final boolean
SERIAL_INTERFACE = true;     private static final int DATA_MASK_OUT_PORT = 0x1F;
private static final int WR_MASK_OUT_PORT = 0x40; // Write.
    private static int animationIndex;
private static long timeout;
    public static void main(String[] args) {
        HAL.init();
        SerialEmitter.init();
init();
    }
    public static void init() {
clear();
    }
    public static void clear() {
showNumber(CLEAR_DISPLAY);
    }
    private static void showNumberParallel(int number) {
        HAL.writeBits(DATA_MASK_OUT_PORT, number);
        HAL.setBits(WR_MASK_OUT_PORT);
        HAL.clrBits(WR_MASK_OUT_PORT);
    }
    private static void showNumberSerial(int number) {
        SerialEmitter.send(SerialEmitter.Destination.RDisplay, number);
    }
    public static void showNumber(int number) {
if (SERIAL_INTERFACE) {
showNumberSerial(number);
        } else {
            showNumberParallel(number);
        }
}
    public static void startAnimation() {
animationIndex = 0;
        showNumber(ANIMATION[animationIndex]);
        timeout = Time.getTimeInMillis() + ANIMATION_TIMEOUT_MS;
    }
    public static void animation() {         if
(timeout < Time.getTimeInMillis()) {
            animationIndex = animationIndex < ANIMATION_MAX_INDEX ? animationIndex +
1 : 0;
            showNumber(ANIMATION[animationIndex]);
timeout += ANIMATION_TIMEOUT_MS;
        }
    }
}
```

Código Java da classe *M*

```
// Maintenance. public class M {     private static final int
M_MASK_IN_PORT = 0x20; // Maintenance.
    public static boolean check() {
return HAL.isBit(M_MASK_IN_PORT);
    }
}
```

Código Java da classe *RouletteGame – App*

```
import isel.leic.utils.Time;
 import
java.util.ArrayList; import
java.util.Random;

public class APP {
    private static final int SQUARE_1_CELL_NUMBER = 0;
private static final int[] SQUARE_1 = {
            0b00000,
            0b11111,
            0b10001,
            0b10101,
            0b10001,
            0b11111,
            0b00000,
0b00000
    };
    private static final int SQUARE_2_CELL_NUMBER = 1;
private static final int[] SQUARE_2 = {
            0b00000,
            0b11111,
            0b10101,
            0b10001,
            0b10101,
            0b11111,
            0b00000,
0b00000
    };
    private static final int SQUARE_3_CELL_NUMBER = 2;
private static final int[] SQUARE_3 = {
            0b00000,
            0b11111,
            0b10011,
            0b10101,
            0b11001,
            0b11111,
            0b00000,
0b00000
    };
    private static final String STATISTICS_FILE_NAME = "Statistics.txt";
private static final String COUNT_FILE_NAME = "Count.txt";     private
static final int MAX_BALANCE = 99;     private static final int
MAX_BETS_PER_NUMBER = 9;
    private static final String MAINTENANCE_OPTIONS[] = {"0-Stats #-Count", "*-Play
8-ShutD"};
    private static final int MAINTENANCE_OPTIONS_MAX_INDEX =
MAINTENANCE_OPTIONS.length - 1;
    private static final int RD_ANIMATION_1_TIMEOUT_MS = 5500; // Go through the
roulette display's external segments.
    private static final int RD_ANIMATION_2_TIMEOUT_MS = 750; // Show random numbers
on the roulette display.
```

```
    private static final int RD_ANIMATION_3_TIMEOUT_MS = 500; // Blink the drawn
number on the roulette display.
    private static final int MAINTENANCE_MODE_OPTIONS_TIMEOUT_MS = 2500;
private static final int GENERIC_TIMEOUT_MS = 5000;     private static
int balance;
    private static int maintenanceOptionsIndex;
private static boolean betMade;     private
static int betNumbers[];
    private static ArrayList<Statistic> statistics;
private static Count count;     public static void
main(String[] args) {          init();          for
(;;) {             printMainMenu();
            while (!('*' == TUI.getKeyKBD() && balance > 0)) {
                if (CoinAcceptor.check()) {
CoinAcceptor.accept();
incrementBalance(1);
count.incrementCoins(1);                    printBalance();
                }
                if (M.check()) {
maintenanceMode();
printMainMenu();
                }
}
gameMode();
save();
        }
}
    private static void init() {
        HAL.init();
        KBD.init();
        SerialEmitter.init();
        LCD.init();
        RouletteDisplay.init();
        TUI.init();
        TUI.setCustomCharLCD(SQUARE_1_CELL_NUMBER, SQUARE_1);
        TUI.setCustomCharLCD(SQUARE_2_CELL_NUMBER, SQUARE_2);
        TUI.setCustomCharLCD(SQUARE_3_CELL_NUMBER, SQUARE_3);
ArrayList<String> fileContent;          statistics = new
ArrayList<>();          for (int i = 0; i < 10; i++) {
statistics.add(new Statistic(i));
        }
        fileContent = FileAccess.read(STATISTICS_FILE_NAME);
if (null != fileContent) {
loadStatistics(fileContent);
        }
        count = new Count();
        fileContent = FileAccess.read(COUNT_FILE_NAME);
if (null != fileContent) {
loadCount(fileContent);
        }
}
    private static void printMainMenu() {
        TUI.clearLCD();
        TUI.printAlignedStringLCD(1, TUI.Alignment.CENTER, "Roulette Game");
        TUI.printAlignedStringLCD(2, TUI.Alignment.LEFT,
```

```
                " 1 " + (char) SQUARE_1_CELL_NUMBER
                + " 2 " + (char) SQUARE_2_CELL_NUMBER
+ " 3 " + (char) SQUARE_3_CELL_NUMBER);
printBalance();
    }
    private static void incrementBalance(int inc) {
if (balance + inc <= MAX_BALANCE) {
balance += inc;
        } else {
            balance = MAX_BALANCE;
        }
}
    private static void printBalance() {
        TUI.printAlignedStringLCD(2, TUI.Alignment.RIGHT, " $" + balance);
    }
    private static void maintenanceMode() {
char key;         while (M.check()) {
TUI.clearLCD();
            TUI.printAlignedStringLCD(1, TUI.Alignment.CENTER, "On Maintenance");
            TUI.printAlignedStringLCD(2, TUI.Alignment.CENTER,
MAINTENANCE_OPTIONS[maintenanceOptionsIndex]);
nextMaintenanceOption();              if
(TUI.NONE != (key =
TUI.waitKeyKBD(MAINTENANCE_MODE_OPTIONS_TIMEOUT_MS))) {
if ('0' == key) { // Statistics.
                    TUI.clearLCD();
                    Statistic s1 = statistics.get(0), s2 = statistics.get(1);
printStatistic(1, s1);                      printStatistic(2, s2);
                    while (TUI.NONE != (key = TUI.waitKeyKBD(GENERIC_TIMEOUT_MS))) {
if ('2' == key) {
                            if (0 != s1.getNumber()) {
s2 = s1;
                                s1 = statistics.get(s2.getNumber() - 1);
printStatistic(1, s1);                                  printStatistic(2,
s2);
                            }
                        } else if ('8' == key) {
if (9 != s2.getNumber()) {
s1 = s2;
                                s2 = statistics.get(s1.getNumber() + 1);
printStatistic(1, s1);                                  printStatistic(2,
s2);
                            }
                        }
                    }
                } else if ('#' == key) { // Count.
                    TUI.clearLCD();
                    TUI.printAlignedStringLCD(1, TUI.Alignment.LEFT, "Games:" +
count.getGames());
                    TUI.printAlignedStringLCD(2, TUI.Alignment.LEFT, "Coins:" +
count.getCoins());
                    TUI.waitKeyKBD(GENERIC_TIMEOUT_MS);
                } else if ('*' == key) { // Play.
                    int balanceBak = balance;
```

```
                    ArrayList<Statistic> statisticsBak = new ArrayList<>();
for (int i = 0; i < 10; i++) {
                        statisticsBak.add(new Statistic(statistics.get(i)));
}
                    Count countBak = new Count(count);
balance = MAX_BALANCE;
                    gameMode();
balance = balanceBak;
statistics = statisticsBak;
count = countBak;
                } else if ('8' == key) { // Shut Down.
                    TUI.clearLCD();
                    TUI.printAlignedStringLCD(1, TUI.Alignment.CENTER, "Shut Down");
TUI.printAlignedStringLCD(2, TUI.Alignment.CENTER, "5-Yes OtherNo");
                    if ('5' == TUI.waitKeyKBD(GENERIC_TIMEOUT_MS)) {
save();                             System.exit(0);
                    }
                }
            }
        }
}
    private static void nextMaintenanceOption() {
        if (MAINTENANCE_OPTIONS_MAX_INDEX == maintenanceOptionsIndex) {
maintenanceOptionsIndex = 0;
        } else {
            maintenanceOptionsIndex++;
        }
}
    private static void printStatistic(int line, Statistic s) {
        TUI.printAlignedStringLCD(line, TUI.Alignment.LEFT, s.getNumber() + ": -> "
+ s.getCount() + " $:" + s.getAmount());
    }
    private static void gameMode() {
        TUI.clearLCD();
        TUI.printAlignedStringLCD(2, TUI.Alignment.LEFT, "0123456789");
printBalance();         betMade = false;         betNumbers = new
int[10];         long timeout = 0;         char key;
        while (0 == timeout || Time.getTimeInMillis() < timeout) {
key = TUI.getKeyKBD();             if (timeout > 0) {
                RouletteDisplay.animation();
            } else if ('#' == key && betMade) {
RouletteDisplay.startAnimation();
                timeout = Time.getTimeInMillis() + RD_ANIMATION_1_TIMEOUT_MS;
            }
            if (key >= '0' && key <= '9'
                    && betNumbers[key - '0'] < MAX_BETS_PER_NUMBER
                    && balance > 0) {
bet(key - '0');
            }
        }
        Random random = new Random();
        randomNumbersAnimation(random, 4, RD_ANIMATION_2_TIMEOUT_MS);
int randomNumber = random.nextInt(10);
RouletteDisplay.showNumber(randomNumber);         if
(betNumbers[randomNumber] > 0) { // Win.
```

```
            int winVal = betNumbers[randomNumber] << 1; // Times 2.
incrementBalance(winVal);
            statistics.get(randomNumber).incrementAmount(winVal);
TUI.printAlignedStringLCD(1, TUI.Alignment.RIGHT, "W$" + winVal);
        } else { // Lose.
int loseVal = 0;
            for (int i = 0; i < 10; i++) {
loseVal += betNumbers[i];
            }
            TUI.printAlignedStringLCD(1, TUI.Alignment.RIGHT, "L$" + loseVal);
        }
        blink(10, RD_ANIMATION_3_TIMEOUT_MS, randomNumber);
statistics.get(randomNumber).incrementCount(1);
count.incrementGames(1);          RouletteDisplay.clear();
    }
    private static void bet(int i) {
betNumbers[i]++;          betMade =
true;          balance--;
        TUI.printCharLCD(1, i + 1, (char) (betNumbers[i] + '0'));
printBalance();
    }
    private static void randomNumbersAnimation(Random random, int count, long
initialTimeout) {          int randomNumber;
        for (int i = 0; i < count; i++) {
randomNumber = random.nextInt(10);
            RouletteDisplay.showNumber(randomNumber);
            Time.sleep(initialTimeout);
initialTimeout *= 1.5f;
        }
}
    private static void blink(int count, long timeout, int number) {
for (int i = 0; i < count; i++) {
            RouletteDisplay.clear();
            Time.sleep(timeout);
            RouletteDisplay.showNumber(number);
            Time.sleep(timeout);
        }
}
    private static void loadStatistics(ArrayList<String> fileContent) {
        String values[];          Statistic
statistic;          for (int i = 0; i < 10; i++) {
values = fileContent.get(i).split(";");
statistic = statistics.get(i);
            statistic.setCount(Integer.valueOf(values[1]));
statistic.setAmount(Integer.valueOf(values[2]));
        }
}
    private static void loadCount(ArrayList<String> fileContent) {
count.setGames(Integer.valueOf(fileContent.get(0)));
count.setCoins(Integer.valueOf(fileContent.get(1)));
    }
    private static void save() {
        ArrayList<String> fileContent = new ArrayList<>();
for (Statistic s : statistics) {
```

```
            fileContent.add(s.getNumber() + ";" + s.getCount() + ";" +
s.getAmount());
        }
        FileAccess.write(STATISTICS_FILE_NAME, fileContent);
fileContent = new ArrayList<>();
        fileContent.add(String.valueOf(count.getGames()));
fileContent.add(String.valueOf(count.getCoins()));
        FileAccess.write(COUNT_FILE_NAME, fileContent);
    }
}
```

Código Java da classe *SerialEmitter*

```
public class SerialEmitter {     public enum Destination {RDisplay,
LCD}     private static final int SS_MASK_OUT_PORT = 0x01; // SOC
Select.     private static final int SCLK_MASK_OUT_PORT = 0x02; //
SOC Clock.     private static final int SDX_MASK_OUT_PORT = 0x04; //
Send.     private static final int DATA_SIZE_BITS = 5;     public
static void main(String[] args) {          HAL.init();
init();     }
    public static void init() {
        HAL.clrBits(SS_MASK_OUT_PORT);
    }
    public static void send(Destination addr, int data) {
HAL.setBits(SS_MASK_OUT_PORT);
sendBit(Destination.LCD == addr);          int bit;          boolean
parity = false;
        for (int i = 0; i < DATA_SIZE_BITS; i++) {
bit = data & 1;              data >>= 1;
parity = (0 == bit) == parity;
sendBit(1 == bit);
}
        sendBit(parity);
        HAL.clrBits(SS_MASK_OUT_PORT);
    }
    private static void sendBit(boolean condition) {
if (condition) {
            HAL.setBits(SDX_MASK_OUT_PORT);          }
else {
            HAL.clrBits(SDX_MASK_OUT_PORT);
        }
        HAL.setBits(SCLK_MASK_OUT_PORT);
        HAL.clrBits(SCLK_MASK_OUT_PORT);
    }
}
```

Código Java da classe *COINACCEPTOR*

```
public class CoinAcceptor {     private static final int
C_MASK_IN_PORT = 0x40; // Coin.     private static final int
A_MASK_OUT_PORT = 0x40; // Accept.
    public static boolean check() {
return HAL.isBit(C_MASK_IN_PORT);
    }
    public static void accept() {
HAL.setBits(A_MASK_OUT_PORT);          while
(HAL.isBit(C_MASK_IN_PORT));
        HAL.clrBits(A_MASK_OUT_PORT);
    }
}
```

Código Java da classe *COUNT*

```
public class Count {
    private int games, coins;
    public Count() {
    }
    public Count(Count count) {
        games = count.getGames();
        coins = count.getCoins();
    }
    public int getGames() {
        return games;
    }
    public int getCoins() {
        return coins;
    }
    public void setGames(int games) {
        this.games = games;
    }
    public void incrementGames(int inc) {
        games += inc;
    }
    public void setCoins(int coins) {
        this.coins = coins;
    }
    public void incrementCoins(int inc) {
        coins += inc;
    }
}
```