ISEL
INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Licenciatura Engenharia Informática e Multimédia

Fundamentos de Sistemas Operativos

Trabalho Prático 3

Docentes:

Jorge Pais

Carlos Gonçalves

Shun Wang Nº45410

João Cunha Nº45412

Arman Freitas Nº45414

Data de entrega:5/12/2018

Índice

Índice de Figuras

Índice de Código

1. Capítulo de introdução

O seguinte trabalho prático visa a continuação do desenvolvimento de uma aplicação multitarefas, que controla um Robot Lego. Esta aplicação contém várias tarefas.

Primeiramente a GUI, onde se pode ativar e desativar outras tarefas, como por exemplo: o comportamento Fugir, o Vaguear e o Evitar.

Como se pôde verificar nos trabalhos propostos anteriormente, o comportamento Fugir, foge de qualquer objeto que o persiga, com uma determinada velocidade, dependente da distância do objeto. Já o Vaguear, faz com que o robot tenha um movimento completamente aleatório e, assim, vagueie pelo meio em que está. E por fim, o Evitar, que evita qualquer objeto que obstrua o caminho.

No entanto, o objetivo do presente trabalho prático, é criar uma nova tarefa, o RobotPlayer. Desta vez, apenas o RobotPlayer "conhecerá" o Robot Lego.

O propósito do RobotPlayer, é a opção de poder gravar qualquer trajetória realizada e, reproduzi-la. Assim, é possível gravar uma trajetória à escolha do utilizador.

A gravação é então guardada num ficheiro externo, para que mais tarde se possa aceder a este mesmo mesmo ficheiro e assim, seja possível reproduzir a trajetória outra vez.

A escrita de cada gravação para o ficheiro externo é feita através de Input/Output Streams em Java.

Uma Stream é definida por uma sequência de dados/informação. Quando um ficheiro é editado, este é guardado na memória, para que assim seja possível que o ficheiro seja modificado. Desta forma, garante-se que nenhum dado seja perdido e/ou o ficheiro acabe danificado.

Existem dois tipos de Streams:

- InputStream

Um InputStream é utilizado para ler dados de uma fonte.

- OutputStream

Um OutputStream é utilizado para escrever dados num destino.

Por fim, para terminar este capítulo, foi-nos também proposto desenvolver uma nova funcionalidade na GUI (fornecida no primeiro trabalho prático). Esta funcionalidade tem o propósito guardar a maioria das definições da GUI para que, quando aberta de novo, esta seja automaticamente atualizada. Esta funcionalidade foi também implementada com a ajuda de Input/OutputStreams.

2. Diagrama de Atividades – RobotPlayer

Como foi explicado no capítulo de introdução, a classe *RobotPlayer* tem o intuito de permitir que o utilizador possa gravar trajetórias com o robot. O utilizador pode também reproduzir uma gravação, encontrada num ficheiro externo.

Desta forma, o diagrama de atividades do *RobotPlayer*, demonstrado na figura seguinte, possui um estado de espera. Este estado, apenas espera que alguma das seguintes atividades seja posta em ação:

- Reproduzir

Esta atividade reproduz a trajetória presente no ficheiro externo.

- Voltar

O Voltar, apenas reproduz a mesma trajetória inversamente, de modo a chegar ao início do percurso (local onde foi reproduzido).

- Gravar

O estado Gravar, grava a trajetória que o robot está a fazer até que o utilizador a pare de gravar.

O diagrama de atividades desta tarefa está presente na seguinte figura:

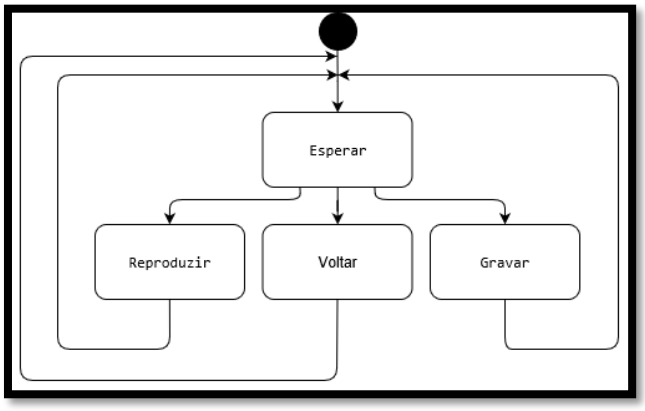

Figura 1 Diagrama de Atividades - RobotPlayer

3. Diagrama de classes em UML da aplicação

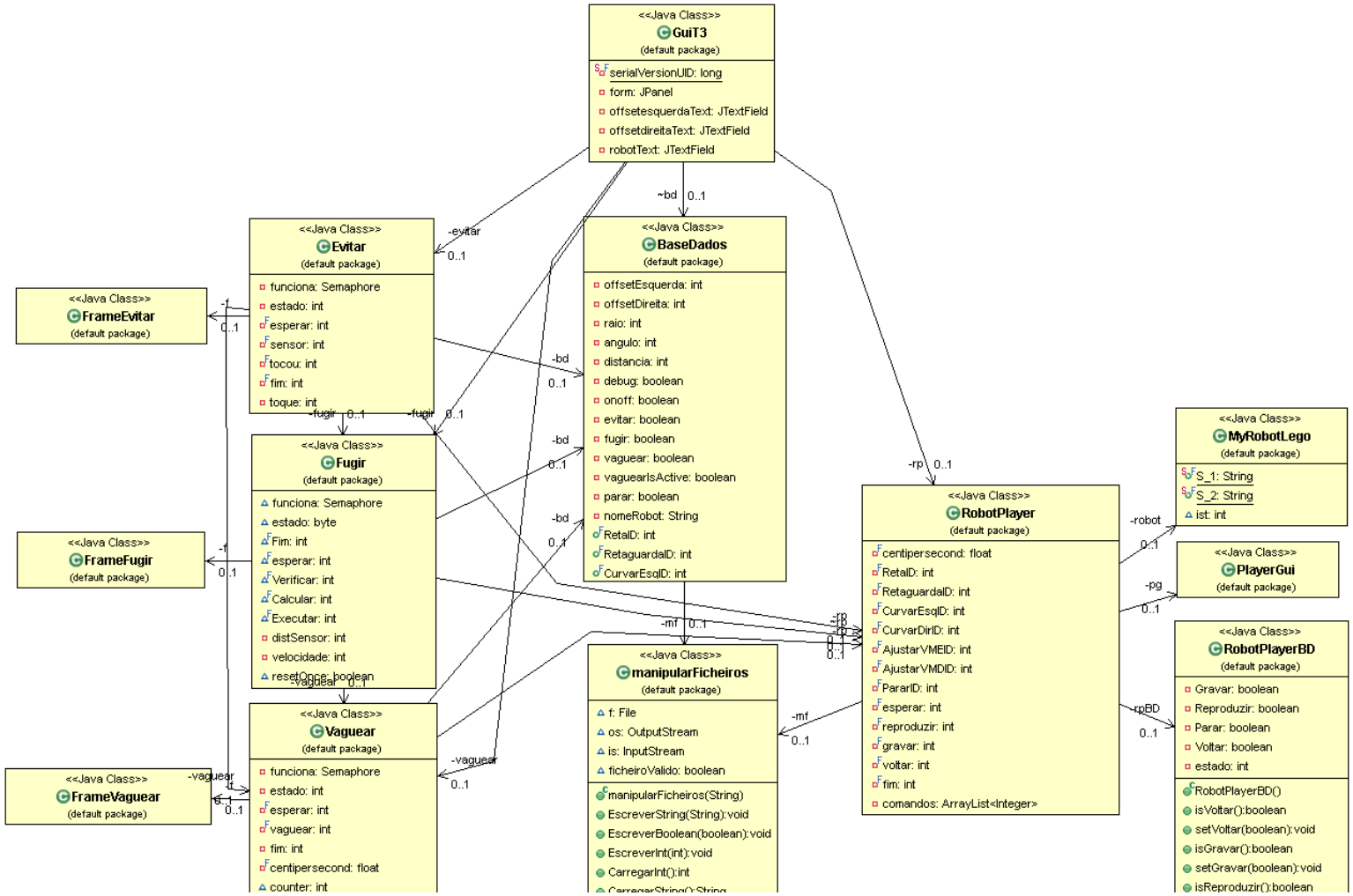

Figura 2 - Diagrama ULM

Data de entrega:5/12/2018

4. Gui do RobotPlayer

A Thread *RobotPlayer* é possuidora de uma GUI, para que o utilizador possa controlar o estado em que esta classe está (Reproduzir, Voltar ou Gravar). Para que o método *run()* da Thread *RobotPlayer* não interfira com o método *run()* de *JFrame*, foi criada uma nova classe (com a ajuda do Windows Builder), a classe *PlayerGui*, que está presente numa instância no *RobotPlayer* e, é inicializada no seu construtor.

PlayerGUI, como o nome sugere, é um *JFrame*, que comunica com a classe *RobotPlayer*, de forma a enviar informação relevante do utilizador, por exemplo se o utilizador deseja gravar a trajetória do robot, ou mesmo reproduzi-la.

De forma a estas duas classes comunicarem entre si (*PlayerGui* e *RobotPlayer*), foi construída ainda outra classe, *RobotPlayerBD*. Esta apenas guarda valores de estado. Assim, estes valores afetam os estados presentes na classe *RobotPlayer*.

A GUI do *RobotPlayer* ficou então com a seguinte apresentação:

Figura 3 Gui - RobotPlayer

Data de entrega:5/12/2018

5. Classe para a manipulação do ficheiro

Como explicado anteriormente, no capítulo de introdução, de modo a conseguirmos ler e escrever ficheiros, utilizámos Input/OutputStreams.

A classe com a finalidade de manipular ficheiros, tem o nome de *manipularFicheiros* e, está contida numa instância, no *RobotPlayer*.

Esta classe possuí diversos métodos para garantir que seja possível fazer todas as alterações necessárias ao ficheiro. Antes de tudo, são criadas as variáveis para criar os InputStreams e OutputStreams e, o ficheiro em si:

```
File f;
OutputStream os;
InputStream is;
```

Código 1 - Criação das variáveis na classe *manipularFicheiros*

Assim, *os*, corresponde ao Stream que nos permite escrever no ficheiro e, *is*, permite-nos ler do ficheiro em questão.

Antes de qualquer leitura ou alteração no ficheiro, é necessário abrir um "canal de comunicação", isto é possível através dos métodos: *osOpen()* e, *isOpen()*, que criam um canal para o OutputStream e InputStream, respetivamente.

De seguida, encontra-se um exemplo da abertura de um canal, presente na classe *manipularFicheiros*:

```
public boolean osOpen() {
	try {
		os = new FileOutputStream(f);
		return true;
	} catch (FileNotFoundException e) {
		e.printStackTrace();
		return false;
	}
}
```

Código 2 - Abertura do canal de escrita (OutputStream)

Após a abertura de um dos canais de comunicação, seja para escrever, seja para ler do ficheiro, é possível escrever ou ler do ficheiro respetivamente. Assim, se o canal de leitura estiver aberto é possível utilizar métodos como o *CarregarComandos()*, que carrega os comandos presentes no ficheiro para um array de inteiros, como mostra a seguinte ilustração:

```
public int[] CarregarComandos() {
	String val = "";
	try {
		while (is.available() > 0) {
			int numericValue = Character.getNumericValue(is.read());
			val+=numericValue;
		}
	} catch (IOException e) {
		e.printStackTrace();
	}
	String[] value = val.split("-1");

	int[] intvalue = new int[value.length];
	for(int i = 0 ; i < intvalue.length ; i++) {
		intvalue[i] = Integer.parseInt(value[i]);
	}
	return intvalue;
}
```

Código 3 - Método para carregar comandos do ficheiro

Por outro lado, se for o canal de escrita que esteja aberto, é possível utilizar métodos de escrita, como é por exemplo o *EscreverString()*, que escreve uma String no ficheiro. Este método está demonstrado no trecho de código seguinte:

```
public void EscreverString(String s) {
	try {
		String c = new String(s);
		c += "=";
		System.out.println(Arrays.toString(c.getBytes()));
		os.write(c.getBytes());
	} catch (IOException e) {
		e.printStackTrace();
	}
}
```

Código 4 - Método para escrever uma String no ficheiro

É de notar que se está a adicionar um *"="* depois da String de entrada, de forma a que mais tarde seja possível distinguir os diferentes valores enviados para o ficheiro, como é o caso do método demonstrado anteriormente, *CarregarComandos()*, que lê cada instrução separada por um caractere *"="*. Este caractere tem apenas uso para distinguir intruções e nada mais.

6. Classe para a manipulação da informação de um comando para o robot

Seguidamente, no desenvolvimento do trabalho, foi construída a classe *RobotPlayer*, que recebe várias ordens da GUI, Vaguear, Fugir e, do Evitar em relação ao movimento a ser realizado pelo Robot Lego.

Como foi referido no capítulo de introdução, apenas o *RobotPlayer* "conhece" a instância do robot, fazendo com que, apenas esta classe tenha acesso direto ao robot.

A classe *RobotPlayer*, envia a informação do movimento para a classe *manipularFicheiros* e para a classe *RobotLegoNXT*, conforme a atividade em questão. Se estiver a fazer *Reta()* e não estiver, nem a reproduzir, nem a gravar, não é enviada qualquer informacão para o *manipularFicheiros*.

No entanto, quando o *manipularFicheiros* recebe informação e o *RobotPlayer* está em estado de reprodução, são guardados os comandos de movimento contidos no ficheiro (guardados num array) e, por fim, o Robot Lego executa o comando.

Relembrado agora a forma de ler os comandos explicada resumidamente no capítulo anterior, cada número escrito no ficheiro encontra-se separado pelo caractere *"="* e, cada número tem o seu significado. Os seus significados podem ser:

- Id, que distinge o comando (Reta, Curva, etc.)
- Tempo de espera (*Thread.sleep()*)
- Um/dois valor(es) da distância, raio e ângulo

```
public RobotPlayer(String nome) {
	// robot = new RobotLegoEV3();
	robot = new RobotLegoNXT();
	// robot = new MyRobotLego();
	rpBD = new RobotPlayerBD();
	pg = new PlayerGui(rpBD);
	mf = new manipularFicheiros(nome);
	comandos = new ArrayList<Integer>();
	rpBD.setEstado(esperar);
}
```

Código 5 - Construtor do *RobotPlayer*

Para exemplificar, temos o método *Curva()*, que realiza uma curva. Apenas se o botão de gravar estiver ativo, a informação relativa as caraterísticas da curva são enviadas para a classe *manipularFicheiros*:

```
	public void Curva(int id, int radious, int angle) {
		if (id == CurvarDirID)
			robot.CurvarDireita(radious, angle);
		else if (id == CurvarEsqID)
			robot.CurvarEsquerda(radious, angle);

		if (rpBD.isGravar()) {
			comandos.add(id);
			comandos.add(radious);
			comandos.add(angle);
			comandos.add(calcularCurvaTempo(radious, angle));
			System.out.println("Curvar : " + id + " " + radious + " " + angle + "com
este sleep: "
					+ calcularCurvaTempo(radious, angle));
		}
	}
```

Código 6 – Método para o robot curvar, na classe *RobotPlayer*

A classe *RobotPlayerBD* é utilizada de modo a guardar informação relevante do utilizador, ou seja, dos butões da GUI do *RobotPlayer* que está situada em *PlayerGui*.

Isto, torna possível a interação do utilizador com o *RobotPlayer* de um modo mais intuitivo e, criando assim uma comunicação entre estas duas classes.

Para a realização do trajeto pedido no enunciado pelo Docente, foi criado o seguinte método:

```
	public void escrevercomandos() {
		Curva(CurvarDirID, 9, 45);
		Curva(CurvarEsqID, 9, 45);
		Reta(RetaID, 10);
		Curva(CurvarEsqID, 9, 45);
		Reta(RetaID, 5);
		Curva(CurvarDirID, 9, 90);
		Curva(CurvarDirID, 9, 30);
		Curva(CurvarEsqID, 9, 30);
		Reta(RetaID, 10);
		Reta(RetaID, 15);
		Parar(false);
	}
```

Código 7 - Método para escrever os comandos pedidos pelo Docente

7. Fim do comportamento do RobotPlayer de forma controlada

Como todas a tarefas, o RobotPlayer tem de ser ligado pelo utilizador na GUI, quando se encontra inativo qualquer ação feita na GUI do RobotPlayer é ignorada e é enviado uma mensagem na Textfield para o informar o utilizador que o RobotPlayer se encontra desligado.

```
public void actionPerformed(ActionEvent arg0) {
	if (!bd.isRobotPlayer()) {
		myPrint("Ligar RobotPlayer");
		bd.setRobotPlayer(!bd.isRobotPlayer());
		rp.setPlayerGui(true);
		rp.setEstado(0);
	} else {
		myPrint("Desligar RobotPlayer");
		bd.setRobotPlayer(!bd.isRobotPlayer());
		rp.setPlayerGui(false);
		rp.setEstado(-1);
	}
	chckbxRobotPlayer.setSelected(bd.isRobotPlayer());
}
```

Código 8 - Ativar/Desativar o RobotPlayer

Quando na GUI é clicado na checkbox do RobotPlayer o estado do RobotPlayer passa a ser esperar, neste estado este fica a espera de comandos ou seja inputs dos butões feitos pelo utilizador, quando é clicado denovo na checkbox o estado passa a ser fim, neste estado a tarefa reage aos inputs do utilizador, simplesmente envia uma mensagem para a Textfield a dizer que a tarefa precisa de estar ligada para fazer ações.

```
public void actionPerformed(ActionEvent e) {
	if (rpBD.getEstado() == esperar) {
		LogTextField.setText("Começar a Reproduzir");
		rpBD.setReproduzir(true);
		rpBD.setEstado(reproduzir);
	}else {
		LogTextField.setText("Ligue o RobotPlayer");
		}
}
```

Código 9 - Exemplo de deteção da inatividade da tarefa (ligada/desligada)

8. Tratamento de exceções efetuado de forma correta

Uma exceção (ou evento excecional), é um problema que surge durante a execução de um programa. Quando existe a ocorrência de uma exceção, o fluxo normal do programa é interrompido e, o programa (ou tarefa) é finalizado de forma anormal. Esta forma de terminar o programa (ou tarefa) não é recomendada, portanto, essas exceções devem ser tratadas.

Neste trabalho, essas exceções são tratadas através do bloco try/catch. O bloco try/catch está presente em várias linguagens de programação (C#, Java, Javascript, Python, etc.).

Todo o código presente neste bloco está "protegido". Se for detetado algum erro, a tarefa não será interrompida de forma anormal.

As exceções de cada tarefa podem ser visualizadas nas "TextFields" das mesmas, facilitando assim o debug ou visualização dos locais de erros na execução do código.

```
try {
	Thread.sleep(100 + (1000*(long) (distancia/centipersecond)));
} catch (InterruptedException e) {
	EnDis.setText("Excepção ao tentar dormir a tarefa ao fazer Reta, erro
	da excepção: " + e.getMessage());
}
```

Código 10 - Exemplo de uma exceção a ser tratada (GUI)

```
try {
	s.acquire();
} catch (InterruptedException e1) {
	myPrint("Excepção ao tentar adquirir o semaforo no Ajuste VMD"
	+ ", erro da excepção: " + e1.getMessage());
}
```

Código 11 - Exemplo de uma exceção a ser tratada (Vaguear)

```
try {
        funciona.acquire(); EnDis.setText("Enabled");
} catch (InterruptedException e) {
        EnDis.setText("Thread do Evitar não está ativa, erro de excepção: " +
        e.getMessage());
}
```

Código 12 - Exemplo de uma exceção a ser tratada (Evitar)

9. Conclusões

Com o propósito de desenvolver uma aplicação multitarefas, onde várias classes estão a ser executadas em "simultâneo", achamos ter chegado às respostas que procurávamos.

A utilização de semáforos foi deveras, importantíssimo para a sincronização das várias tarefas a correr, pois todas estas estão a ser executadas "simultaneamente".

O desenvolvimento do *RobotPlayer* foi significativo para realizar as várias ordens pretendidas pelo utilizador, por exemplo: gravar um novo percurso, carregar um percurso antigo ou mesmo voltar ao ponto inicial depois de fazer um percurso.

Desta forma permitimos que o utilizador realize muitas operações, o que cria uma grande liberdade e fluidez na utilização da aplicação. Para tal, foi necessário manipular o ficheiro de modo a guardar informações novas e, ler dados antigos.

Assim, este projeto permitiu-nos ter um grande desenvolvimento de conhecimentos sobre manipulação de ficheiros (através de Input/OutputStreams) e, mesmo colocá-los em prática.

Em boa verdade, conseguimos reconhecer o "poder" de uma aplicação multitarefas. Compreendemos como funciona cada Thread em si, de modo a conseguirmos colocar o nosso próprio projeto a funcionar. Apesar de muitos impasses que ocorreram ao longo do desenvolvimento, conseguimos chegar aos objetivos pretendidos.

10. Bibliografia

[1] Folhas de FSO versão 2, Jorge Pais, 2018

11. Anexos

11.1 – Classe BaseDados

```
public class BaseDados {

	private int offsetEsquerda, offsetDireita, raio, angulo, distancia;
	private boolean debug, onoff, evitar, fugir, vaguear, vaguearIsActive,
parar, robotPlayer;
	private String nomeRobot;
	private manipularFicheiros mf;

	public final int RetaID = 0;
	public final int RetaguardaID = 1;
	public final int CurvarEsqID = 2;
	public final int CurvarDirID = 3;
	public final int AjustarVME = 3;
	public final int AjustarVMD = 4;

	public BaseDados() {
		offsetDireita = offsetEsquerda = raio = angulo = distancia = 0;
		debug = true;
		onoff = false;
		evitar = false;
		vaguear = false;
		fugir = false;
		parar = false;
		robotPlayer = false;
		vaguearIsActive = false;
		nomeRobot = new String("");

	}

	public void ImportarGUI() {
		mf = new manipularFicheiros(nomeRobot + ".dat");
		if (mf.isOpen()) {
			setDebug(mf.CarregarBoolean());
			setOffsetEsquerda(mf.CarregarInt());
			setOffsetDireita(mf.CarregarInt());
			setNomeRobot(mf.CarregarString());
			setRaio(mf.CarregarInt());
			setAngulo(mf.CarregarInt());
			setDistancia(mf.CarregarInt());
			setOnoff(mf.CarregarBoolean());
			mf.isClose();
		}
	}

	public void EscreverGUI() {
		mf = new manipularFicheiros(nomeRobot + ".dat");
		if (mf.osOpen()) {
			EscreverBoolean(debug);
			EscreverInt(offsetEsquerda);
			EscreverInt(offsetDireita);
			EscreverString(nomeRobot);
			EscreverInt(raio);
			EscreverInt(angulo);
			EscreverInt(distancia);
```

```
                EscreverBoolean(onoff);
                System.out.println("Gravação Terminada");
                mf.osClose();
        }
}

private void EscreverString(String s) {
        mf.EscreverString(s);
}

private void EscreverInt(int i) {
        mf.EscreverInt(i);
}

private void EscreverBoolean(boolean b) {
        mf.EscreverBoolean(b);
}

public boolean isVaguearIsActive() {
        return vaguearIsActive;
}

public void setVaguearIsActive(boolean vaguearIsActive) {
        this.vaguearIsActive = vaguearIsActive;
}

public boolean isParar() {
        return parar;
}

public void setParar(boolean p) {
        parar = p;
}

public int getRaio() {
        return raio;
}

public void setRaio(int raio) {
        this.raio = raio;
}

public int getAngulo() {
        return angulo;
}

public void setAngulo(int angulo) {
        this.angulo = angulo;
}

public int getDistancia() {
        return distancia;
}

public void setDistancia(int distancia) {
        this.distancia = distancia;
}

public String getNomeRobot() {
```

```
        return nomeRobot;
    }

    public void setNomeRobot(String nomeRobot) {
        this.nomeRobot = nomeRobot;
    }

    public boolean isDebug() {
        return debug;
    }

    public void setDebug(boolean debug) {
        this.debug = debug;
    }

    public boolean isOnoff() {
        return onoff;
    }

    public void setOnoff(boolean onoff) {
        this.onoff = onoff;
    }

    public int getOffsetEsquerda() {
        return offsetEsquerda;
    }

    public void setOffsetEsquerda(int offsetEsquerda) {
        this.offsetEsquerda = offsetEsquerda;
    }

    public int getOffsetDireita() {
        return offsetDireita;
    }

    public void setOffsetDireita(int offsetDireita) {
        this.offsetDireita = offsetDireita;
    }

    public boolean isEvitar() {
        return evitar;
    }

    public void setEvitar(boolean evitar) {
        this.evitar = evitar;
    }

    public boolean isFugir() {
        return fugir;
    }

    public void setFugir(boolean fugir) {
        this.fugir = fugir;
    }

    public boolean isVaguear() {
        return vaguear;
    }
```

```
	public void setVaguear(boolean vaguear) {
		this.vaguear = vaguear;
	}

	public int getRetaID() {
		return RetaID;
	}

	public int getRetaguardaID() {
		return RetaguardaID;
	}

	public int getCurvarEsqID() {
		return CurvarEsqID;
	}

	public int getCurvarDirID() {
		return CurvarDirID;
	}

	public int getAjustarVME() {
		return AjustarVME;
	}

	public int getAjustarVMD() {
		return AjustarVMD;
	}

	public boolean isRobotPlayer() {
		return robotPlayer;
	}

	public void setRobotPlayer(boolean robotPlayer) {
		this.robotPlayer = robotPlayer;
	}

}
```

11.2 – Classe Evitar

```
import java.util.concurrent.Semaphore;


public class Evitar implements Runnable {

	private RobotPlayer rp;
	private Fugir fugir;
	private Vaguear vaguear;
	private FrameEvitar f;
	private Semaphore funciona;
	private BaseDados bd;

	private int estado;
	private final int esperar = 0;
	private final int sensor = 1;
	private final int tocou = 2;
	private final int fim = -1;
```

```
	private int toque;

	public Evitar (BaseDados bd, RobotPlayer rp, Fugir fugir, Vaguear
vaguear) {
		this.bd = bd;
		this.rp = rp;
		this.fugir = fugir;
		this.vaguear = vaguear;

		// Definir Variaveis
		this.toque = 0;
		this.funciona = new Semaphore (0);
		estado = esperar;
		f = new FrameEvitar();
	}

	//	public Evitar (Semaphore s, MyRobotLego robot) {
	//
	//		// Definir Variaveis
	//		this.toque = 0;
	//		this.funciona = new Semaphore (0);
	//		this.s = s;
	//		this.robot = robot;
	//
	//		estado = esperar;
	//		milisecondspermeter = 6438;
	//
	//		// Criar GUI
	//		setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
	//		setBounds(100, 100, 575, 180);
	//
	//		contentPane = new JPanel();
	//		setContentPane(contentPane);
	//		contentPane.setLayout(null);
	//
	//		JLabel Vaguearlb = new JLabel("Evitar");
	//		Vaguearlb.setBounds(10, 0, 100, 50);
	//		contentPane.add(Vaguearlb);
	//
	//		EnDis = new JTextField ("Disabled");
	//		EnDis.setBounds(10, 100, 535, 22);
	//		contentPane.add(EnDis);
	//		EnDis.setColumns(10);
	//		setVisible(true);
	//
	//		EvitarTextField = new JTextField();
	//		EvitarTextField.setBounds(10, 70, 535, 22);
	//		contentPane.add(EvitarTextField);
	//		EvitarTextField.setColumns(10);
	//		setVisible(true);
	//
	//	}

	@Override
	public void run() {

		// Evitar Loop
		System.out.println("Tarefa: Evitar a correr...");
		while (estado != fim) {
```

```
                        Automato ();
                }
        }

        private void Automato () {
                switch (estado) {
                case esperar:
                        try { funciona.acquire();
f.writeEvitarTextField("Enabled");
                        } catch (InterruptedException e) { f.writeEnDis("Thread
do Evitar não está ativa, erro de excepção: " + e.getMessage());}

                        estado = sensor;

                        break;

                case sensor:
                        toque = rp.SensorToque(1);
                        if (toque == 1) {
                                if(bd.isVaguear())
                                        vaguear.Deactivate();

                                if(bd.isFugir())
                                        fugir.Deactivate();


                                f.writeEnDis("Tocou: 1");
                                estado = tocou;
                                break;
                        } else {
                                f.writeEnDis("Não Tocou");
                                try { Thread.sleep(250);
                                } catch (InterruptedException e) {
f.writeEnDis("Excepção ao tentar dormir a tarefa na leitura do SensorToque,
erro da excepção: " + e.getMessage());}
                                break;
                        }


                case tocou:
                        rp.Parar(true);
                        rp.SetVelocidade(50);
                        rp.Reta(bd.getRetaID(), -15);
                        rp.Parar(false);
                        rp.Curva(bd.getCurvarEsqID(), 0, 90);
                        rp.Parar(false);

                        try { Thread.sleep(2000);
                        } catch (InterruptedException e) {f.writeEnDis("Excepção
ao tentar dormir a tarefa depois do Evitar, erro da excepção: " +
e.getMessage());}


                        if(bd.isVaguear())
                                vaguear.Activate();


                        if(bd.isFugir())
                                fugir.Activate();
```

```
                        estado = sensor;
                        break;

                default:
                        break;
                }

        }


        public void Activate () {
                funciona.release();
        }

        public void Deactivate () {
                funciona.drainPermits();
                estado = esperar;
        }

        public void Terminate() {
                estado = fim;
                funciona.release();
        }
}
```

11.3 – Classe Frame Evitar

```
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTextField;

public class FrameEvitar extends JFrame{
        private static final long serialVersionUID = 1L;
        private JPanel contentPane;
        private JTextField EnDis;
        private JTextField EvitarTextField;
        public FrameEvitar() {
                // Criar GUI
                setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                setBounds(100, 100, 575, 180);

                contentPane = new JPanel();
                setContentPane(contentPane);
                contentPane.setLayout(null);

                JLabel Evitarlb = new JLabel("Evitar");
                Evitarlb.setBounds(10, 0, 100, 50);
                contentPane.add(Evitarlb);

                EnDis = new JTextField ();
                EnDis.setBounds(10, 100, 535, 22);
                contentPane.add(EnDis);
                EnDis.setColumns(10);
                setVisible(true);
```

```
		EvitarTextField = new JTextField();
		EvitarTextField.setBounds(10, 70, 535, 22);
		contentPane.add(EvitarTextField);
		EvitarTextField.setColumns(10);
		setVisible(true);
	}

	public void writeEnDis(String s) {
		EnDis.setText(s);
	}
	public void writeEvitarTextField(String s) {
		EvitarTextField.setText(s);
	}

}
```

11.4 – Classe Fugir

```
import java.util.concurrent.Semaphore;

public class Fugir implements Runnable {

	RobotPlayer rp;
	Semaphore funciona;
	private Vaguear vaguear;
	private BaseDados bd;

	byte estado;
	final int Fim = -1;
	final int esperar = 0;
	final int Verificar = 1;
	final int Calcular = 2;
	final int Executar = 3;

	private int distSensor, velocidade;

	boolean resetOnce = false;

	private FrameFugir f;
	public Fugir(BaseDados bd, RobotPlayer rp, Vaguear vaguear) {
		this.bd = bd;
		this.rp = rp;
		this.vaguear = vaguear;
		estado = esperar;
		funciona = new Semaphore(0);
		f = new FrameFugir();

	}

	//	public Fugir(MyRobotLego robot, Semaphore s) {
	//		this.s = s;
	//		estado = Esperar;
	//		haTrabalho = new Semaphore(0);
	//		this.robot = robot;
	//		tempoParado = 0;
	//		estadoAnterior = Esperar;
	//	}
```

```
	@Override
	public void run() {

		// Evitar Loop
		System.out.println("Tarefa: Fugir a correr...");

		while (estado != Fim) {
			executafugir ();
		}
	}


	public void executafugir() {

		switch (estado) {
		case esperar:
			try { funciona.acquire();
f.writeFugirTextField("Enabled");
			} catch (InterruptedException e1) { f.writeEnDis("Thread
do Fugir não está ativa, erro de excepção: " + e1.getMessage()); }

			estado = Verificar;
			break;

		case Verificar:
			distSensor = rp.SensorUS(3);

			if(distSensor < 60 && estado == Verificar) {
				resetOnce = true;
				if(bd.isVaguear())
					vaguear.Deactivate();

				estado = Calcular;
			}

			else {
				try { Thread.sleep(250); } catch
(InterruptedException e) {  f.writeEnDis("Excepção ao tentar dormir a tarefa
quando o sensorUS é >60, erro da excepção: " + e.getMessage());}
				if(resetOnce) {
					rp.Parar(true);
					rp.SetVelocidade(50);
					resetOnce = false;
				}
				if(bd.isVaguear()) {
					rp.Parar(true);
					vaguear.Activate();
				}
			}
			break;

		case Calcular:
			velocidade = (-10/6)*distSensor +100;
			estado = Executar;
			break;


		case Executar:
```

```
                        rp.SetVelocidade((int)velocidade);
                        rp.Reta(bd.getRetaID(),1);
                        estado = Verificar;
                        break;


                }

        }
        public void Deactivate() {
                funciona.drainPermits();
                estado = esperar;
        }

        public void Activate() {
                funciona.release();
        }

        public void Terminate() {
                estado = Fim;
                funciona.release();
        }



}
```

11.5 – Classe FrameFugir

```
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTextField;

public class FrameFugir extends JFrame{
        private static final long serialVersionUID = 1L;
        private JPanel contentPane;
        private JTextField EnDis;
        private JTextField FugirTextField;

        public FrameFugir() {

                // Criar GUI
                setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                setBounds(100, 100, 575, 180);

                contentPane = new JPanel();
                setContentPane(contentPane);
                contentPane.setLayout(null);

                JLabel Fugirlb = new JLabel("Fugir");
                Fugirlb.setBounds(10, 0, 100, 50);
                contentPane.add(Fugirlb);

                EnDis = new JTextField ();
                EnDis.setBounds(10, 100, 535, 22);
```

```
            contentPane.add(EnDis);
            EnDis.setColumns(10);
            setVisible(true);

            FugirTextField = new JTextField();
            FugirTextField.setBounds(10, 70, 535, 22);
            contentPane.add(FugirTextField);
            FugirTextField.setColumns(10);
            setVisible(true);
      }

      public void writeEnDis(String s) {
            EnDis.setText(s);
      }

      public void writeFugirTextField(String s) {
            FugirTextField.setText(s);
      }

}
```

11.6 – Classe Vaguear

```
import java.util.concurrent.Semaphore;

public class Vaguear implements Runnable {

      private RobotPlayer rp;
      private BaseDados bd;
      private FrameVaguear f;
      private Semaphore funciona;

      private int estado;
      private final int esperar = 0;
      private final int vaguear = 1;
      private int fim = -1;
      private final float centipersecond = 15.98f;

      public Vaguear(BaseDados bd, RobotPlayer rp) {
            // Definir variaveis
            this.bd = bd;
            this.rp = rp;
            this.estado = esperar;
            this.funciona = new Semaphore(0);
            f = new FrameVaguear();

      }
      //
      // public Vaguear (Semaphore s, MyRobotLego robot) {
      // // Definir variaveis
      // this.estado = esperar;
      // this.s = s;
      // this.robot = robot;
      // this.centipersecond = 15.53f;
      // this.funciona = new Semaphore (0);
      //
```

```
        // // Criar GUI
        // setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        // setBounds(100, 100, 575, 180);
        //
        // contentPane = new JPanel();
        // setContentPane(contentPane);
        // contentPane.setLayout(null);
        //
        // JLabel Vaguearlb = new JLabel("Vaguear");
        // Vaguearlb.setBounds(10, 0, 100, 50);
        // contentPane.add(Vaguearlb);
        //
        // EnDis = new JTextField ("Disabled");
        // EnDis.setBounds(10, 100, 535, 22);
        // contentPane.add(EnDis);
        // EnDis.setColumns(10);
        // setVisible(true);
        //
        // VaguearTextField = new JTextField();
        // VaguearTextField.setBounds(10, 70, 535, 22);
        // contentPane.add(VaguearTextField);
        // VaguearTextField.setColumns(10);
        // setVisible(true);
        //
        // }

        int counter = 0;

        @Override
        public void run() {
                // Vaguear Loop

                System.out.println("Tarefa: Vaguear a correr...");
                while (estado != fim) {
                        Automato();
                }
        }

        private void Automato() {
                switch (estado) {
                case esperar:
                        try {
                                funciona.acquire();
                                f.writeVaguearTextField("Enabled");
                        } catch (InterruptedException e1) {
                                f.writeEnDis("Thread do Vaguear não está ativa,
erro de excepção: " + e1.getMessage());
                        }

                        if (estado == esperar) {
                                estado = vaguear;
                        }
                        break;

                case vaguear:
                        vaguearTrajeto();
                        counter++;
                        if (counter == 16) {
                                rp.Parar(true);
```

```
                        Deactivate();
                        estado = esperar;
                        counter = 0;
                    }
                    break;
                }
            }

            private void vaguearTrajeto() {

                int valor = (int) (Math.random() * 3);
                switch (valor) {

                case 0:
                    int distancia = (int) (Math.random() * (50 - 10) + 10);
                    f.writeEnDis("Distancia: " + distancia);
                    rp.Reta(bd.getRetaID(), distancia);
                    System.out.println("reta sleep: " + (int) ((distancia /
centipersecond) * 1000));
                    try {
                        Thread.sleep((int) ((distancia / centipersecond) *
1000));
                    } catch (InterruptedException e) {
                        f.writeEnDis("Excepção ao tentar dormir a tarefa ao
fazer Reta, erro da excepção: " + e.getMessage());
                    }
                    break;

                // case 1:
                // rp.Parar(true);
                // f.writeEnDis("Parar");
                // try { Thread.sleep(2000);
                // }catch (InterruptedException e) { f.writeEnDis("Excepção ao
tentar dormir a
                // tarefa parar, erro da excepção: " + e.getMessage()); }
                // break;

                case 1:
                    int raio = (int) (Math.random() * (30 - 10) + 10);
                    int ang = (int) (Math.random() * (90 - 20) + 20);
                    f.writeEnDis("Curvar Direita-> Raio: " + raio + ",
Angulo: " + ang);
                    rp.Curva(bd.getCurvarDirID(), raio, ang);
                    System.out
                                .println("curvar direita sleep: " + (int)
(((raio * ang * (Math.PI / 2)) / centipersecond) * 1000));
                    try {
                        Thread.sleep((int) (((raio * ang * (Math.PI / 180))
/ centipersecond) * 1000));
                    } catch (InterruptedException e) {
                        f.writeEnDis("Excepção ao tentar dormir a tarefa ao
fazer curva à direita, erro da excepção: "
                                + e.getMessage());
                    }
                    break;

                case 2:
                    int r = (int) (Math.random() * (30 - 10) + 10);
                    int angulo = (int) (Math.random() * (90 - 20) + 20);
```

```
                    rp.Curva(bd.getCurvarEsqID(), r, angulo);
                    f.writeEnDis("Curvar Esquerda-> Raio: " + r + ", Angulo:
" + angulo);
                    System.out.println(
                                "curvar esquerda sleep: " + (int) (((r *
angulo * (Math.PI / 2)) / centipersecond) * 1000));
                    try {
                          Thread.sleep((int) (((r * angulo * (Math.PI / 180))
/ centipersecond) * 1000));
                    } catch (InterruptedException e) {
                          f.writeEnDis("Excepção ao tentar dormir a tarefa ao
fazer curva à esquerda, erro da excepção: "
                                      + e.getMessage());
                    }
                    break;

              default:
                    System.out.println("ERROR");
                    break;
              }
        }

        public void Activate() {
              funciona.release();

        }

        public void Deactivate() {
              funciona.drainPermits();
              estado = esperar;

        }

        public void Terminate() {
              estado = fim;
              funciona.release();

        }
}
```

11.7 – Classe FrameVaguear

```
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTextField;

public class FrameVaguear extends JFrame{
        private static final long serialVersionUID = 1L;
        private JPanel contentPane;
        private JTextField EnDis;
        private JTextField VaguearTextField;

        public FrameVaguear() {

              setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
              setBounds(100, 100, 575, 180);
```

```
            contentPane = new JPanel();
            setContentPane(contentPane);
            setLayout(null);

            JLabel Vaguearlb = new JLabel("Vaguear");
            Vaguearlb.setBounds(10, 0, 100, 50);
            add(Vaguearlb);

            EnDis = new JTextField ("Enable/Disable");
            EnDis.setBounds(10, 100, 535, 22);
            contentPane.add(EnDis);
            EnDis.setColumns(10);
            setVisible(true);

            VaguearTextField = new JTextField();
            VaguearTextField.setBounds(10, 70, 535, 22);
            contentPane.add(VaguearTextField);
            VaguearTextField.setColumns(10);
            setVisible(true);

      }

      public void writeEnDis(String s) {
            EnDis.setText(s);
      }
      public void writeVaguearTextField(String s) {
            VaguearTextField.setText(s);
      }


}
```

11.8 – Classe GUIT3

```
import java.awt.Color;
import java.awt.EventQueue;
import java.awt.Font;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.File;
import java.util.concurrent.Semaphore;

import javax.swing.JButton;
import javax.swing.JCheckBox;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JRadioButton;
import javax.swing.JTextField;
import javax.swing.UIManager;
import javax.swing.border.EmptyBorder;

public class GuiT3 extends JFrame {
```

```
private static final long serialVersionUID = 1L;

BaseDados bd;
private JPanel form;
private JTextField offsetesquerdaText;
private JTextField offsetdireitaText;
private JTextField robotText;
private JTextField raioText;
private JTextField anguloText;
private JTextField distanciaText;
private JTextField logText;
private JCheckBox debugCheck;
private int nPermits;
private Semaphore s;
private Evitar evitar;
private Vaguear vaguear;
private Fugir fugir;
private RobotPlayer rp;
private File f;

/*
 * Launch the application.
 */
public static void main(String[] args) {
	EventQueue.invokeLater(new Runnable() {
		public void run() {
			try {
				GuiT3 frame = new GuiT3();
				frame.setVisible(true);
			} catch (Exception e) {
				e.printStackTrace();
			}
		}
	});
}

/*
 * Create the frame.
 */
public GuiT3() {
	nPermits = 1;
	s = new Semaphore(nPermits);
	// robot = new RobotLegoNXT ();

	bd = new BaseDados();

	setResizable(false);
	setTitle(bd.getNomeRobot());
	setBackground(Color.WHITE);
	setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
	setBounds(0, 9, 640, 480);
	form = new JPanel();
	form.setToolTipText("");
	form.setBackground(Color.ORANGE);
	form.setBorder(new EmptyBorder(5, 5, 5, 5));
	setContentPane(form);
	form.setLayout(null);

	offsetesquerdaText = new JTextField();
```

```
		offsetesquerdaText.addActionListener(new ActionListener() {
			public void actionPerformed(ActionEvent e) {
				if (offsetdireitaText.getText().equals(""))
					bd.setOffsetEsquerda(0);
				else

	bd.setOffsetEsquerda(Integer.parseInt(offsetesquerdaText.getText()));

				try {
					s.acquire();
				} catch (InterruptedException e1) {
					myPrint("Excepção ao tentar adquirir o
semaforo no ajuste VME" + ", erro da excepção: "
							+ e1.getMessage());
				}

				rp.AjustarVM(bd.CurvarEsqID,
bd.getOffsetEsquerda());

				s.release();

				myPrint("Offset Esquerda " + bd.getOffsetEsquerda()
+ "");

			}
		});

		JCheckBox evitarCheck = new JCheckBox("Evitar");
		evitarCheck.addActionListener(new ActionListener() {
			public void actionPerformed(ActionEvent arg0) {
				if (bd.isEvitar()) {
					myPrint("Parar Evitar");
					bd.setEvitar(!bd.isEvitar());
					evitar.Deactivate();
					rp.Parar(true);
				}

				else {
					myPrint("Ligar Evitar");
					bd.setEvitar(!bd.isEvitar());
					evitar.Activate();

				}
				evitarCheck.setSelected(bd.isEvitar());
			}
		});

		JCheckBox fugirCheck = new JCheckBox("Fugir");
		fugirCheck.setForeground(Color.BLACK);
		fugirCheck.addActionListener(new ActionListener() {

			public void actionPerformed(ActionEvent e) {
				if (bd.isFugir()) {
					myPrint("Desligar Evitar");
					bd.setFugir(!bd.isFugir());
					fugir.Deactivate();
					rp.Parar(true);
				}
```

```
                else {
                        myPrint("Ligar Evitar");
                        bd.setFugir(!bd.isFugir());
                        fugir.Activate();

                }
                fugirCheck.setSelected(bd.isFugir());
        }
});

JCheckBox vaguearCheck = new JCheckBox("Vaguear");

vaguearCheck.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent arg0) {
                if (bd.isVaguear()) {
                        myPrint("Desligar Vaguear");
                        bd.setVaguear(!bd.isVaguear());
                        vaguear.Deactivate();
                        rp.Parar(true);
                }

                else {
                        myPrint("Ligar Vaguear");
                        bd.setVaguear(!bd.isVaguear());
                        vaguear.Activate();

                }
                vaguearCheck.setSelected(bd.isVaguear());
        }
});
vaguearCheck.setBackground(Color.ORANGE);
vaguearCheck.setForeground(Color.BLACK);
vaguearCheck.setBounds(513, 138, 92, 23);
form.add(vaguearCheck);
fugirCheck.setBackground(Color.ORANGE);
fugirCheck.setBounds(513, 234, 92, 23);
form.add(fugirCheck);
evitarCheck.setForeground(Color.BLACK);
evitarCheck.setBackground(Color.ORANGE);
evitarCheck.setBounds(513, 185, 92, 23);
form.add(evitarCheck);

JLabel comportamento = new JLabel("Comportamentos");
comportamento.setBackground(Color.ORANGE);
comportamento.setFont(new Font("Lucida Bright", Font.PLAIN,
14));
comportamento.setBounds(475, 70, 130, 39);
form.add(comportamento);
offsetesquerdaText.setBackground(Color.WHITE);

offsetesquerdaText.setBounds(11, 39, 52, 20);
form.add(offsetesquerdaText);
offsetesquerdaText.setColumns(10);

JLabel offsetesquerda = new JLabel("Left Offset");
offsetesquerda.setFont(new Font("Lucida Bright", Font.PLAIN,
14));
offsetesquerda.setBounds(11, 9, 77, 20);
form.add(offsetesquerda);
```

```
		JLabel offsetdireita = new JLabel("Right Offset");
		offsetdireita.setFont(new Font("Lucida Bright", Font.PLAIN,
14));
		offsetdireita.setBounds(538, 9, 86, 20);

		form.add(offsetdireita);

		offsetdireitaText = new JTextField();
		offsetdireitaText.addActionListener(new ActionListener() {
			public void actionPerformed(ActionEvent e) {
				if (offsetdireitaText.getText().equals(""))
					bd.setOffsetDireita(0);
				else

	bd.setOffsetDireita(Integer.parseInt(offsetdireitaText.getText()));

				try {
					s.acquire();
				} catch (InterruptedException e1) {
					myPrint("Excepção ao tentar adquirir o
semaforo no Ajuste VMD" + ", erro da excepção: "
							+ e1.getMessage());
				}

				rp.AjustarVM(bd.AjustarVMD, bd.getOffsetDireita());

				s.release();
				myPrint("Offset Direita " + bd.getOffsetDireita() +
"");
			}
		});
		offsetdireitaText.setBackground(Color.WHITE);
		offsetdireitaText.setColumns(10);
		offsetdireitaText.setBounds(538, 39, 52, 20);
		form.add(offsetdireitaText);

		JLabel robotname = new JLabel("Name");
		robotname.setFont(new Font("Lucida Bright", Font.PLAIN, 18));
		robotname.setBounds(11, 82, 52, 14);
		form.add(robotname);

		robotText = new JTextField();
		robotText.setText(bd.getNomeRobot());
		robotText.addActionListener(new ActionListener() {
			public void actionPerformed(ActionEvent e) {

				bd.setNomeRobot(robotText.getText());

				if (robotText.getText().equals("")) {
					System.out.println("Introduza um nome");
					return;
				}

				f = new File(robotText.getText() + ".dat");
				if (f.exists()) {
					System.out.println("Nome do robot existe nos
ficheiros, importando configurações do: " + f.getName());
					bd.ImportarGUI();
```

```
                                    debugCheck.setSelected(bd.isDebug());

        offsetesquerdaText.setText(String.valueOf(bd.getOffsetEsquerda()));

        offsetdireitaText.setText(String.valueOf(bd.getOffsetDireita()));

        raioText.setText(String.valueOf(bd.getRaio()));

        anguloText.setText(String.valueOf(bd.getAngulo()));

        distanciaText.setText(String.valueOf(bd.getDistancia()));
                                    robotText.setText(bd.getNomeRobot());
                              } else {
                                    System.out.println("Não foi encontrado robot
com este nome, por favor guarde as configurações");
                              }
                              myPrint("Nome do Robot: " + bd.getNomeRobot() +
"");

                        }
                  });
                  robotText.setBounds(11, 107, 200, 20);
                  form.add(robotText);
                  robotText.setColumns(10);

                  JLabel lblRaio = new JLabel("Raio");
                  lblRaio.setFont(new Font("Lucida Bright", Font.PLAIN, 14));
                  lblRaio.setBounds(418, 9, 30, 20);
                  form.add(lblRaio);

                  JLabel angulo = new JLabel("Angle");
                  angulo.setFont(new Font("Lucida Bright", Font.PLAIN, 14));
                  angulo.setBounds(287, 9, 39, 20);
                  form.add(angulo);

                  JLabel lblDist = new JLabel("Dist\u00E2ncia");
                  lblDist.setFont(new Font("Lucida Bright", Font.PLAIN, 14));
                  lblDist.setBounds(145, 9, 70, 20);
                  form.add(lblDist);

                  raioText = new JTextField();
                  raioText.addActionListener(new ActionListener() {
                        public void actionPerformed(ActionEvent e) {
                              if (raioText.getText().equals(""))
                                    bd.setRaio(0);
                              else

        bd.setRaio(Integer.parseInt(raioText.getText()));
                              myPrint("Raio: " + bd.getRaio() + "");

                        }
                  });
                  raioText.setBounds(418, 39, 52, 20);
                  form.add(raioText);
                  raioText.setColumns(10);

                  anguloText = new JTextField();
                  anguloText.addActionListener(new ActionListener() {
                        public void actionPerformed(ActionEvent e) {
```

```
                        if (anguloText.getText().equals(""))
                              bd.setAngulo(0);
                        else

      bd.setAngulo(Integer.parseInt(anguloText.getText()));
                        myPrint("Angulo: " + bd.getAngulo() + "");

                  }
            });
            anguloText.setColumns(10);
            anguloText.setBounds(287, 39, 52, 20);
            form.add(anguloText);

            distanciaText = new JTextField();
            distanciaText.addActionListener(new ActionListener() {
                  public void actionPerformed(ActionEvent e) {
                        if (distanciaText.getText().equals(""))
                              bd.setDistancia(0);
                        else

      bd.setDistancia(Integer.parseInt(distanciaText.getText()));
                        myPrint("Distância: " + bd.getDistancia() + "");

                  }
            });
            distanciaText.setColumns(10);
            distanciaText.setBounds(145, 39, 52, 20);
            form.add(distanciaText);

            JButton frenteBtn = new JButton("\u25B2 " + "Frente");
            frenteBtn.addActionListener(new ActionListener() {
                  public void actionPerformed(ActionEvent e) {

                        try {
                              s.acquire();
                        } catch (InterruptedException e1) {
                              myPrint("Excepção ao tentar adquirir o
semaforo na Reta" + ", erro da excepção: "
                                          + e1.getMessage());
                        }

                        rp.Reta(bd.RetaID, bd.getDistancia());
                        rp.Parar(false);
                        s.release();
                        myPrint("Andar para frente: " + bd.getDistancia() +
"");

                  }
            });
            frenteBtn.setBackground(Color.RED);
            frenteBtn.setForeground(Color.BLACK);
            frenteBtn.setBounds(169, 160, 103, 30);
            form.add(frenteBtn);

            JButton esquerdaBtn = new JButton("\u25C4 " + "Esquerda");
            esquerdaBtn.addActionListener(new ActionListener() {
                  public void actionPerformed(ActionEvent e) {

                        try {
```

```
                        s.acquire();
                    } catch (InterruptedException e1) {
                        myPrint("Excepção ao tentar adquirir o
semaforo na curva à esquerda" + ", erro da excepção: "
                                + e1.getMessage());
                    }

                    rp.Curva(bd.CurvarEsqID, bd.getRaio(),
bd.getAngulo());
                    rp.Parar(false);

                    s.release();
                    myPrint("Curvar para esquerda: " + bd.getRaio() + "
" + bd.getAngulo() + " ");

                }
            });
            esquerdaBtn.setForeground(Color.BLACK);
            esquerdaBtn.setBackground(Color.GREEN);
            esquerdaBtn.setBounds(56, 201, 103, 30);
            form.add(esquerdaBtn);

            JButton direitaBtn = new JButton("\u25BA " + "Direita");
            direitaBtn.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent e) {

                    try {
                        s.acquire();
                    } catch (InterruptedException e1) {
                        myPrint("Excepção ao tentar adquirir o
semaforo na curva à direita" + ", erro da excepção: "
                                + e1.getMessage());
                    }

                    rp.Curva(bd.CurvarDirID, bd.getRaio(),
bd.getAngulo());
                    rp.Parar(false);
                    s.release();
                    myPrint("Curvar para direita: " + bd.getRaio() + "
" + bd.getAngulo() + " ");

                }
            });
            direitaBtn.setForeground(Color.BLACK);
            direitaBtn.setBackground(Color.BLUE);
            direitaBtn.setBounds(283, 201, 103, 30);
            form.add(direitaBtn);

            JButton retaguardaBtn = new JButton("\u25BC " + "Retaguarda");
            retaguardaBtn.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent e) {

                    try {
                        s.acquire();
                    } catch (InterruptedException e1) {
                        myPrint("Excepção ao tentar adquirir o
semaforo na Retaguarda" + ", erro da excepção: "
                                + e1.getMessage());
                    }
```

```
                        rp.Reta(bd.RetaguardaID, bd.getDistancia() * -1);
                        rp.Parar(false);

                        s.release();
                        myPrint("Andar para retaguarda: " +
bd.getDistancia() + "");

                  }
            });
            retaguardaBtn.setForeground(Color.BLACK);
            retaguardaBtn.setBackground(Color.YELLOW);
            retaguardaBtn.setBounds(169, 242, 103, 30);
            form.add(retaguardaBtn);

            JButton pararBtn = new JButton("\u26d4 " + "Parar");
            pararBtn.addActionListener(new ActionListener() {
                  public void actionPerformed(ActionEvent e) {

                        try {
                              s.acquire();
                        } catch (InterruptedException e1) {
                              myPrint("Excepção ao tentar adquirir o
semaforo no Parar" + ", erro da excepção: "
                                          + e1.getMessage());
                        }
                        rp.Parar(true);
                        s.release();
                        myPrint("Parar = true");
                  }

            });
            pararBtn.setForeground(Color.WHITE);
            pararBtn.setBackground(Color.BLACK);
            pararBtn.setBounds(169, 201, 103, 30);
            form.add(pararBtn);

            debugCheck = new JCheckBox("Debug");
            debugCheck.setSelected(true);
            debugCheck.addActionListener(new ActionListener() {
                  public void actionPerformed(ActionEvent e) {
                        bd.setDebug(debugCheck.isSelected());
                        myPrint("Debug: " + bd.isDebug());
                  }
            });
            debugCheck.setFont(new Font("Lucida Bright", Font.PLAIN, 14));
            debugCheck.setBackground(Color.ORANGE);
            debugCheck.setBounds(16, 294, 70, 23);
            form.add(debugCheck);

            JLabel log = new JLabel("Log");
            log.setFont(new Font("Lucida Bright", Font.PLAIN, 18));
            log.setBounds(21, 325, 39, 30);
            form.add(log);

            logText = new JTextField();
            logText.setBackground(Color.ORANGE);
            logText.setEditable(false);
            logText.setBounds(21, 356, 399, 70);
```

```
		form.add(logText);
		logText.setColumns(10);

		JRadioButton onoffBtn = new JRadioButton("On/Off");
		onoffBtn.setEnabled(true);
		onoffBtn.addActionListener(new ActionListener() {
			public void actionPerformed(ActionEvent arg0) {
				if (!bd.isOnoff()) {
					myPrint("Ligar Robot");
					bd.setOnoff(!bd.isOnoff());
					rp.OpenNXT(bd.getNomeRobot());
				} else {
					myPrint("Desligar Robot");
					bd.setOnoff(!bd.isOnoff());
					rp.CloseNXT();

				}
				onoffBtn.setSelected(bd.isOnoff());

			}
		});
		onoffBtn.setBackground(Color.ORANGE);
		onoffBtn.setBounds(228, 106, 109, 23);
		form.add(onoffBtn);

		JButton importarGUIbtn = new JButton("ImportarGUI");
		importarGUIbtn.addActionListener(new ActionListener() {
			public void actionPerformed(ActionEvent arg0) {
				bd.ImportarGUI();
				debugCheck.setSelected(bd.isDebug());

offsetesquerdaText.setText(String.valueOf(bd.getOffsetEsquerda()));

offsetdireitaText.setText(String.valueOf(bd.getOffsetDireita()));
				raioText.setText(String.valueOf(bd.getRaio()));
				anguloText.setText(String.valueOf(bd.getAngulo()));

distanciaText.setText(String.valueOf(bd.getDistancia()));
				robotText.setText(bd.getNomeRobot());
			}
		});
		importarGUIbtn.setBounds(487, 403, 103, 23);
		form.add(importarGUIbtn);

		JButton escreverGuibtn = new JButton("EscreverGui");

escreverGuibtn.setBackground(UIManager.getColor("Button.background"));
		escreverGuibtn.addActionListener(new ActionListener() {
			public void actionPerformed(ActionEvent e) {
				bd.EscreverGUI();
			}
		});
		escreverGuibtn.setBounds(487, 369, 103, 23);
		form.add(escreverGuibtn);

		JCheckBox chckbxRobotPlayer = new JCheckBox("Robot Player");
		chckbxRobotPlayer.addActionListener(new ActionListener() {
			public void actionPerformed(ActionEvent arg0) {
				if (!bd.isRobotPlayer()) {
```

```
                                myPrint("Ligar RobotPlayer");
                                bd.setRobotPlayer(!bd.isRobotPlayer());
                                rp.setPlayerGui(true);
                                rp.setEstado(0);
                        } else {
                                myPrint("Desligar RobotPlayer");
                                bd.setRobotPlayer(!bd.isRobotPlayer());
                                rp.setPlayerGui(false);
                                rp.setEstado(-1);
                        }
                        chckbxRobotPlayer.setSelected(bd.isRobotPlayer());
                }
        });
        chckbxRobotPlayer.setForeground(Color.BLACK);
        chckbxRobotPlayer.setBackground(Color.ORANGE);
        chckbxRobotPlayer.setBounds(513, 281, 111, 23);
        form.add(chckbxRobotPlayer);

        new Thread(rp = new RobotPlayer("comandos.dat")).start();
        new Thread(vaguear = new Vaguear(bd, rp)).start();
        new Thread(fugir = new Fugir(bd, rp, vaguear)).start();
        new Thread(evitar = new Evitar(bd, rp, fugir, vaguear)).start();
    }

    private void myPrint(String text) {
        if (bd.isDebug())
            logText.setText(text);

    }
}
```

11.8 – Classe manipularFicheiros

```
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;

public class manipularFicheiros {

    File f;
    OutputStream os;
    InputStream is;
    boolean ficheiroValido;

    public manipularFicheiros(String nomeAbsoluto) {
        f = new File(nomeAbsoluto);
        try {
            f.createNewFile();
            ficheiroValido = true;
        } catch (IOException e) {
            ficheiroValido = false;
        }
```

```
	}

	public void EscreverString(String s) {
		try {
			String c = new String(s);
			c += "=";
			os.write(c.getBytes());
		} catch (IOException e) {
			e.printStackTrace();
		}
	}

	public void EscreverBoolean(boolean b) {
		try {
			String mark = "";
			int value = (byte) (b == true ? 1 : 0);
			mark += String.valueOf(value) + "=";
			os.write(mark.getBytes());
		} catch (IOException e) {
			e.printStackTrace();
		}
	}

	public void EscreverInt(int i) {
		String integer = String.valueOf(i);
		integer += "=";
		try {
			os.write(integer.getBytes());
		} catch (IOException e) {
			e.printStackTrace();
		}

	}

	public int CarregarInt() {
		String number = "";
		try {
			while (is.available() > 0) {
				int numericValue =
Character.getNumericValue(is.read());
				if (numericValue != -1) {
					number += numericValue;
				} else {
					break;
				}
			}
		} catch (IOException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}
		if(number.equals(""))
			return 0;
		return Integer.parseInt(number);

	}

	public String CarregarString() {
		String number = "";
		try {
```

```
                while (is.available() > 0) {
                        char numericValue = (char) is.read();
                        if (numericValue != 61) {
                                number += numericValue;
                        } else {
                                break;
                        }
                }
        } catch (IOException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
        }

        return number;

}

public boolean CarregarBoolean() {
        boolean b = false;
        try {
                while (is.available() > 0) {
                        int numericValue =
Character.getNumericValue(is.read());
                        if (numericValue != -1) {
                                b = (numericValue == 1) ? true : false;
                        } else {
                                break;
                        }
                }
        } catch (IOException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
        }
        return b;
}


public int[] CarregarComandos() {
        String val = "";
        try {
                while (is.available() > 0) {
                        int numericValue =
Character.getNumericValue(is.read());
                        val+=numericValue;
                }
        } catch (IOException e) {
                e.printStackTrace();
        }
        String[] value = val.split("-1");

        int[] intvalue = new int[value.length];
        for(int i = 0 ; i < intvalue.length ; i++) {
                intvalue[i] = Integer.parseInt(value[i]);
        }
        return intvalue;
}

public boolean isFicheiroValido() {
        return ficheiroValido;
```

```
	}

	public boolean osOpen() {
		try {
			os = new FileOutputStream(f);
			return true;
		} catch (FileNotFoundException e) {
			e.printStackTrace();
			return false;
		}
	}

	public void osClose() {
		try {
			os.flush();
			os.close();
		} catch (IOException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}
	}


	public boolean isOpen() {
		try {
			is = new FileInputStream(f);
			return true;
		} catch (FileNotFoundException e) {
			e.printStackTrace();
			return false;
		}
	}

	public void isClose() {
		try {
			is.close();
		} catch (IOException e) {
			e.printStackTrace();
		}
	}

}
```

11.9 – Classe MyRobotLego

```
public class MyRobotLego {
	public static final String S_1 = null;
	public static final String S_2 = null;
	int ist;

	public void OpenNXT (String nome) {
		System.out.println("Open NXT: " + nome);
		try {
			Thread.sleep(1000);
		} catch (InterruptedException e) {
			e.printStackTrace();
		}
	}

	public void Reta(int distancia){
		System.out.println("Reta " + distancia);
		try {
			Thread.sleep(100);
		} catch (InterruptedException e) {
			e.printStackTrace();
		}
	}

	public void Parar(boolean valor)  {
		System.out.println("Parar " + valor);
		try {
			Thread.sleep(100);
		} catch (InterruptedException e) {
			e.printStackTrace();
		}
	}

	public void CurvarEsquerda(int raio, int angulo) {
		System.out.println("Curvar Esquerda " + raio + " " + angulo);
		try {
			Thread.sleep(100);
		} catch (InterruptedException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}
	}

	public void CurvarDireita(int raio, int angulo) {
		System.out.println("Curvar Direita " + raio + " " + angulo);
		try {
			Thread.sleep(100);
		} catch (InterruptedException e) {
			e.printStackTrace();
		}
	}

	public int SensorToque() {
		double mypoint = Math.random()*(100);
		if(mypoint<10) {
			return 1;
		}
```

```
		return 0;
	}

	public boolean OpenEV3(String nome) {
		boolean boo;
		if(nome.equals("EV4")==false)
			boo = false;
		else
			boo = true;
		System.out.println("Ligar " + nome + ": " + boo);
		try {
			Thread.sleep(100);
		} catch (InterruptedException e) {
			e.printStackTrace();
		}
		return boo;
	}

	public void CloseNXT() {
		System.out.println("Desligar");
		try {
			Thread.sleep(100);
		} catch (InterruptedException e) {
			e.printStackTrace();
		}
	}

	public void SetVelocidade(int i) {
		System.out.println("setVelocidade: " + i);
		try {
			Thread.sleep(100);
		} catch (InterruptedException e) {
			e.printStackTrace();
		}
	}

	public int SensorToque(int s1) {
		int var = (Math.random() > .50d) ? 1 : 0;

//		System.out.print("Sensor toque: " + var + "\n");
		return var;
	}
	public int SensorUS(int s) {
		return (int)Math.random()*(255);
	}

	public void AjustarVME(int offsetEsquerda) {
		// TODO Auto-generated method stub

	}

	public void AjustarVMD(int offsetDireita) {
		// TODO Auto-generated method stub

	}
}
```

11.10 – Classe RobotPlayer

```
import java.util.ArrayList;

public class RobotPlayer implements Runnable {
	private final float centipersecond = 15.98f;
	private final int RetaID = 0;
	private final int RetaguardaID = 1;
	private final int CurvarEsqID = 2;
	private final int CurvarDirID = 3;
	private final int AjustarVMEID = 4;
	private final int AjustarVMDID = 5;
	private final int PararID = 6;

	// private RobotLegoNXT robot;
	private MyRobotLego robot;
	// private RobotLegoEV3 robot;
	private final int esperar = 0;
	private final int reproduzir = 1;
	private final int gravar = 2;
	private final int voltar = 3;
	private final int fim = -1;

	private manipularFicheiros mf;
	private PlayerGui pg;
	private RobotPlayerBD rpBD;
	private ArrayList<Integer> comandos;

	public RobotPlayer(String nome) {

		// robot = new RobotLegoEV3();
		// robot = new RobotLegoNXT();
		robot = new MyRobotLego();
		rpBD = new RobotPlayerBD();
		pg = new PlayerGui(rpBD);
		mf = new manipularFicheiros(nome);
		comandos = new ArrayList<Integer>();
		rpBD.setEstado(fim);
	}

	@Override
	public void run() {
		while (rpBD.getEstado() != fim) {
			automato();
		}
		try {
			Thread.sleep(100);
		} catch (InterruptedException e) {
		}

	}

	private void automato() {
		switch (rpBD.getEstado()) {
		case esperar:
			try {
				Thread.sleep(100);
			} catch (InterruptedException e) {
```

```
                }
                break;
            case reproduzir:
                if (mf.isOpen()) {
                    Reproduzir(mf, true);
                    robot.Parar(true);
                    mf.isClose();
                    rpBD.setEstado(esperar);
                } else {
                    System.out.println("Não foi possível ler o
ficheiro");
                }
                break;
            case gravar:
                if (mf.osOpen()) {
//                      escrevercomandos();
                    for (Integer a : comandos) {
                        mf.EscreverInt(a);
                    }
                    rpBD.setGravar(false);
                    mf.osClose();
                } else {
                    System.out.println("Não foi possível escrever no
ficheiro");
                }
                comandos.clear();
                rpBD.setEstado(esperar);
                break;

            case voltar:
                if (mf.isOpen()) {
                    robot.CurvarDireita(0, 180);
                    try {
                        Thread.sleep(2190);
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                    Reproduzir(mf, false);
                    robot.Parar(true);
                    mf.isClose();
                    rpBD.setEstado(esperar);
                }
                break;
            }
        }

        public void escrevercomandos() {
            Curva(CurvarDirID, 9, 45);
            Curva(CurvarEsqID, 9, 45);
            Reta(RetaID, 10);
            Curva(CurvarEsqID, 9, 45);
            Reta(RetaID, 5);
            Curva(CurvarDirID, 9, 90);
            Curva(CurvarDirID, 9, 30);
            Curva(CurvarEsqID, 9, 30);
            Reta(RetaID, 10);
            Reta(RetaID, 15);
            Parar(false);
        }
```

```
	public String[] Voltar(manipularFicheiros manipularf) {
		int msg[] = manipularf.CarregarComandos();
		String msgreversa[] = new String[msg.length];

		for (int i = 0; i < msg.length; i++) {
			switch (msg[i]) {
			case RetaID:
				int value = msg[i + 1];
				int sleep = msg[i + 2];

				for (int j = msgreversa.length - 1; j >= 0; j--) {
					if (msgreversa[j] == null) {
						msgreversa[j] = String.valueOf(sleep);
						msgreversa[j - 1] =
String.valueOf(value);
						msgreversa[j - 2] =
String.valueOf(RetaID);
						break;
					}
				}
				// id - valor - sleep

				i += 2;
				break;
			case RetaguardaID:
				int value1 = msg[i + 1];
				int sleep1 = msg[i + 2];

				for (int j = msgreversa.length - 1; j >= 0; j--) {
					if (msgreversa[j] == null) {
						msgreversa[j] =
String.valueOf(sleep1);
						msgreversa[j - 1] =
String.valueOf(value1);
						msgreversa[j - 2] =
String.valueOf(RetaguardaID);
						break;
					}
				}
				i += 2;
				break;
			case CurvarEsqID:
				int Raio = msg[i + 1];
				int Angulo = msg[i + 2];
				int sleep2 = msg[i + 3];

				for (int j = msgreversa.length - 1; j >= 0; j--) {
					if (msgreversa[j] == null) {
						msgreversa[j] =
String.valueOf(sleep2);
						msgreversa[j - 1] =
String.valueOf(Angulo);
						msgreversa[j - 2] =
String.valueOf(Raio);
						msgreversa[j - 3] =
String.valueOf(CurvarEsqID);
						break;
					}
```

```
                        }

                        i += 3;
                        break;
                  case CurvarDirID:
                        int Raio1 = msg[i + 1];
                        int Angulo1 = msg[i + 2];
                        int sleep3 = msg[i + 3];

                        for (int j = msgreversa.length - 1; j >= 0; j--) {
                              if (msgreversa[j] == null) {
                                    msgreversa[j] =
String.valueOf(sleep3);
                                    msgreversa[j - 1] =
String.valueOf(Angulo1);
                                    msgreversa[j - 2] =
String.valueOf(Raio1);
                                    msgreversa[j - 3] =
String.valueOf(CurvarDirID);
                                    break;
                              }
                        }
                        i += 3;
                        break;
                  case PararID:
                        boolean bool = msg[i + 1] == 1 ? true : false;
                        int sleep4 = msg[i + 2];

                        for (int j = msgreversa.length - 1; j >= 0; j--) {
                              if (msgreversa[j] == null) {
                                    msgreversa[j] =
String.valueOf(sleep4);
                                    msgreversa[j - 1] =
String.valueOf(bool ? 1 : 0);
                                    msgreversa[j - 2] =
String.valueOf(PararID);
                                    break;
                              }
                        }
                        i += 2;
                        break;
                  }
            }
            return msgreversa;
      }

      public int[] convertArray(String value[]) {
            int[] intvalue = new int[value.length];
            for (int i = 0; i < intvalue.length; i++) {
                  intvalue[i] = Integer.parseInt(value[i]);
            }
            return intvalue;
      }

      public void Reproduzir(manipularFicheiros manipularf, boolean t) {
            int msg[];
            if (!t) {
                  String[] v = Voltar(manipularf);
                  msg = convertArray(v);
```

```
		} else
			msg = manipularf.CarregarComandos();

		for (int i = 0; i < msg.length; i++) {
			switch (msg[i]) {
			case RetaID:
				int value = msg[i + 1];
				int sleep = msg[i + 2];
				robot.Reta(value);
				System.out.println("Reta: " + value + " " + sleep);
				try {
					Thread.sleep(sleep);
				} catch (InterruptedException e) {
					pg.setText("Excepção ao tentar dormir a
tarefa ao fazer Reta, erro da excepção: "
							+ e.getMessage());
				}
				i += 2;
				break;
			case RetaguardaID:
				int value1 = msg[i + 1];
				int sleep1 = msg[i + 2];
				robot.Reta(value1);
				System.out.println("RetaGuarda: " + value1 + " " +
sleep1);
				try {
					Thread.sleep(sleep1);
				} catch (InterruptedException e) {
					pg.setText("Excepção ao tentar dormir a
tarefa ao fazer Retaguarda, erro da excepção: "
							+ e.getMessage());
				}
				i += 2;
				break;
			case CurvarEsqID:
				int Raio = msg[i + 1];
				int Angulo = msg[i + 2];
				int sleep2 = msg[i + 3];
				if (!t)
					robot.CurvarDireita(Raio, Angulo);
				else
					robot.CurvarEsquerda(Raio, Angulo);
				System.out.println("CurvarEsquerda: " + Raio + " "
+ Angulo + " " + sleep2);
				try {
					Thread.sleep(sleep2);
				} catch (InterruptedException e) {
					pg.setText("Excepção ao tentar dormir a
tarefa ao fazer curva à esquerda, erro da excepção: "
							+ e.getMessage());
				}

				i += 3;
				break;
			case CurvarDirID:
				int Raio1 = msg[i + 1];
				int Angulo1 = msg[i + 2];
				int sleep3 = msg[i + 3];
				if (!t)
```

```
                                robot.CurvarEsquerda(Raio1, Angulo1);
                        else
                                robot.CurvarDireita(Raio1, Angulo1);
                        System.out.println("CurvarDireita: " + Raio1 + " "
+ Angulo1 + " " + sleep3);
                        try {
                                Thread.sleep(sleep3);
                        } catch (InterruptedException e) {
                                pg.setText("Excepção ao tentar dormir a
tarefa ao fazer curva à direita, erro da excepção: "
                                                + e.getMessage());
                        }
                        i += 3;
                        break;
                case PararID:
                        boolean bool = msg[i + 1] == 1 ? true : false;
                        int sleep4 = msg[i + 2];
                        robot.Parar(bool);
                        try {
                                Thread.sleep(sleep4);
                        } catch (InterruptedException e) {
                                pg.setText("Excepção ao tentar dormir a
tarefa ao fazer Parar, erro da excepção: "
                                                + e.getMessage());
                        }
                        i += 2;
                        break;
                }
        }
        rpBD.setReproduzir(false);
    }

    public void OpenNXT(String nome) {
        robot.OpenNXT(nome);
        // robot.OpenEV3(nome);
    }

    public void CloseNXT() {
        robot.CloseNXT();
        // robot.CloseEV3();
    }

    public int SensorToque(int toque) {
        if (toque == 1) {
                return robot.SensorToque(RobotLegoNXT.S_1);
        } else if (toque == 2) {
                return robot.SensorToque(RobotLegoNXT.S_2);
        } else if (toque == 3) {
                return robot.SensorToque(RobotLegoNXT.S_3);
        } else {
                return robot.SensorToque(RobotLegoNXT.S_4);
        }
    }

    public int SensorUS(int us) {
        if (us == 1) {
                return robot.SensorUS(RobotLegoNXT.S_1);
        } else if (us == 2) {
                return robot.SensorUS(RobotLegoNXT.S_2);
```

```
			} else if (us == 3) {
				return robot.SensorUS(RobotLegoNXT.S_3);
			} else {
				return robot.SensorUS(RobotLegoNXT.S_4);
			}
		}

		public void SetVelocidade(int i) {
			robot.SetVelocidade(i);
		}

		public void Parar(boolean b) {
			robot.Parar(b);
			int c = (b) ? 1 : 0;
			if (rpBD.isGravar()) {
				comandos.add(PararID);
				comandos.add(c);
				if (c == 1) {
					comandos.add(1500);
				} else {
					comandos.add(0);
				}

				//System.out.println("Foi gravado: Parar");
			}
		}

		public void Reta(int id, int value) {
			robot.Reta(value);

			if (rpBD.isGravar()) {
				comandos.add(id);
				comandos.add(value);
				comandos.add(calcularRetaTempo(value));
				//System.out.println("Reta: " + id + " " + value + " " +
"com este sleep: " + calcularRetaTempo(value));
			}

		}

		public void Curva(int id, int radious, int angle) {
			if (id == CurvarDirID)
				robot.CurvarDireita(radious, angle);
			else if (id == CurvarEsqID)
				robot.CurvarEsquerda(radious, angle);

			if (rpBD.isGravar()) {
				comandos.add(id);
				comandos.add(radious);
				comandos.add(angle);
				comandos.add(calcularCurvaTempo(radious, angle));
				//System.out.println("Curvar : " + id + " " + radious + "
" + angle + "com este sleep: "
							//+ calcularCurvaTempo(radious, angle));
			}
		}

		public void AjustarVM(int id, int value) {
			if (id == AjustarVMDID)
```

```
                robot.AjustarVMD(value);

            else if (id == AjustarVMEID)
                robot.AjustarVME(value);
        }

        public int calcularRetaTempo(int valor) {
            return Math.abs((int) ((valor / centipersecond) * 1000));
        }

        public int calcularCurvaTempo(int raio, int angulo) {
            return (int) ((raio * Math.toRadians(angulo) / centipersecond) *
1000);
        }

        public void setEstado(int estado) {
            rpBD.setEstado(estado);
        }

        public int getEstado() {
            return rpBD.getEstado();
        }

        public void setPlayerGui(boolean b) {
            if(!b)
                pg.setOnOff(false);
            else
                pg.setOnOff(true);
        }
}
```

11.11 – Classe RobotPlayerBD

```
public class RobotPlayerBD {

        private boolean Gravar, Reproduzir, Parar, Voltar;
        private int estado;

        public RobotPlayerBD() {
            Reproduzir = Parar = Gravar = Voltar = false;
            estado = 0;
        }

        public boolean isVoltar() {
            return Voltar;
        }

        public void setVoltar(boolean voltar) {
            Voltar = voltar;
        }

        public boolean isGravar() {
            return Gravar;
        }

        public void setGravar(boolean Gravar) {
```

```
		this.Gravar = Gravar;
	}

	public boolean isReproduzir() {
		return Reproduzir;
	}

	public void setReproduzir(boolean Reproduzir) {
		this.Reproduzir = Reproduzir;
	}

	public boolean isParar() {
		return Parar;
	}

	public void setParar(boolean Parar) {
		this.Parar = Parar;
	}

	public int getEstado() {
		return estado;
	}

	public void setEstado(int estado) {
		this.estado = estado;
	}
}
```

11.13 – Classe PlayerGui

```
import java.awt.Color;
import java.awt.Cursor;
import java.awt.Font;
import java.awt.SystemColor;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.awt.event.MouseMotionAdapter;

import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTextField;
import javax.swing.SwingConstants;

public class PlayerGui extends JFrame {

	private static final long serialVersionUID = 1L;
	private final int esperar = 0;
	private final int reproduzir = 1;
	private final int gravar = 2;
	private final int voltar = 3;

	private int xPos;
```

```
private int yPos;
private JPanel contentPane;
private JTextField LogTextField;
private RobotPlayerBD rpBD;
private JLabel label;

public PlayerGui(RobotPlayerBD rpBD) {
	this.rpBD = rpBD;
	xPos = 0;
	yPos = 0;

	setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
	setBounds(100, 100, 746, 400);

	setUndecorated(true);

	contentPane = new JPanel();
	contentPane.setBackground(Color.WHITE);
	setContentPane(contentPane);
	contentPane.setLayout(null);

	JPanel panel = new JPanel();
	panel.setBackground(SystemColor.textHighlight);
	panel.setBounds(0, 0, 746, 52);
	panel.setLayout(null);
	contentPane.add(panel);

	label = new JLabel("Robot Player Desligado");
	label.setForeground(new Color(255, 0, 0));
	label.setHorizontalAlignment(SwingConstants.CENTER);
	label.setFont(new Font("Consolas", Font.PLAIN, 40));
	label.setBounds(0, 0, 746, 62);
	panel.add(label);

	JButton playBtn = new JButton("");
	playBtn.setCursor(new Cursor(Cursor.HAND_CURSOR));
	playBtn.addActionListener(new ActionListener() {
		public void actionPerformed(ActionEvent e) {
			if (rpBD.getEstado() == esperar) {
				LogTextField.setText("Começar a Reproduzir");
				rpBD.setReproduzir(true);
				rpBD.setEstado(reproduzir);
			}else {
				LogTextField.setText("Ligue o RobotPlayer");
			}
		}
	});
	playBtn.setBackground(Color.WHITE);
	playBtn.setIcon(new ImageIcon("playIcon.png"));
	playBtn.setBounds(220, 84, 98, 70);
	contentPane.add(playBtn);

	JButton recordBtn = new JButton("");
	recordBtn.setCursor(new Cursor(Cursor.HAND_CURSOR));
	recordBtn.addActionListener(new ActionListener() {
		public void actionPerformed(ActionEvent e) {
			if (rpBD.getEstado() == esperar) {
				LogTextField.setText("Começar a Gravar");
				rpBD.setGravar(true);
```

```
                        }else {
                                LogTextField.setText("Ligue o RobotPlayer");
                        }
                }
        });
        recordBtn.setBackground(Color.WHITE);
        recordBtn.setIcon(new ImageIcon("recordIcon.png"));
        recordBtn.setBounds(10, 84, 98, 70);
        contentPane.add(recordBtn);

        JButton stopBtn = new JButton("");
        stopBtn.setCursor(new Cursor(Cursor.HAND_CURSOR));
        stopBtn.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent arg0) {
                        if (rpBD.getEstado() == esperar) {
                                if (rpBD.isGravar()) {
                                        LogTextField.setText("Terminar o
Gravar");
                                        rpBD.setEstado(gravar);
                                } else if (rpBD.isReproduzir()) {
                                        LogTextField.setText("Terminar o
Reproduzir");
                                        rpBD.setReproduzir(false);
                                        rpBD.setEstado(esperar);
                                } else {
                                        rpBD.setEstado(esperar);
                                }
                        }else {
                                LogTextField.setText("Ligue o RobotPlayer");
                        }
                }
        });

        stopBtn.setBackground(Color.WHITE);
        stopBtn.setIcon(new ImageIcon("stopIcon.png"));
        stopBtn.setBounds(638, 84, 98, 70);
        contentPane.add(stopBtn);

        LogTextField = new JTextField();
        LogTextField.setEditable(false);
        LogTextField.setBounds(10, 228, 726, 82);
        contentPane.add(LogTextField);
        LogTextField.setColumns(10);

        JLabel lblDebug = new JLabel("Log");
        lblDebug.setFont(new Font("Consolas", Font.PLAIN, 20));
        lblDebug.setBounds(10, 193, 98, 24);
        contentPane.add(lblDebug);

        JLabel frameDrag = new JLabel("");
        frameDrag.addMouseListener(new MouseAdapter() {
                @Override
                public void mousePressed(MouseEvent e) {
                        xPos = e.getX();
                        yPos = e.getY();
                }
        });
        frameDrag.addMouseMotionListener(new MouseMotionAdapter() {
                @Override
```

```
		public void mouseDragged(MouseEvent e) {
			int x = e.getXOnScreen();
			int y = e.getYOnScreen();

			setLocation(x - xPos, y - yPos);
		}
	});
	frameDrag.setBounds(0, 0, 746, 52);
	panel.add(frameDrag);

	JButton offBtn = new JButton("");
	offBtn.setBounds(668, 321, 68, 68);
	contentPane.add(offBtn);
	offBtn.setCursor(new Cursor(Cursor.HAND_CURSOR));
	offBtn.addActionListener(new ActionListener() {
		public void actionPerformed(ActionEvent e) {
			System.exit(0);
		}
	});
	offBtn.setIcon(new ImageIcon("offIcon.png"));
	offBtn.setBackground(Color.WHITE);

	JButton btnNewButton = new JButton("");
	btnNewButton.addActionListener(new ActionListener() {
		public void actionPerformed(ActionEvent arg0) {
			if (rpBD.getEstado() == esperar) {
				rpBD.setVoltar(true);
				rpBD.setEstado(voltar);
				LogTextField.setText("Voltar ao inicio");
			}else {
				LogTextField.setText("Ligue o RobotPlayer");
			}
		}
	});
	btnNewButton.setBackground(Color.WHITE);
	btnNewButton.setIcon(new ImageIcon("refresh.png"));
	btnNewButton.setBounds(434, 84, 98, 70);
	contentPane.add(btnNewButton);

	JLabel lblGravar = new JLabel("Gravar");
	lblGravar.setHorizontalAlignment(SwingConstants.CENTER);
	lblGravar.setVerticalAlignment(SwingConstants.BOTTOM);
	lblGravar.setFont(new Font("Consolas", Font.PLAIN, 20));
	lblGravar.setBounds(10, 158, 98, 24);
	contentPane.add(lblGravar);

	JLabel lblReproduzir = new JLabel("Reproduzir");
	lblReproduzir.setHorizontalAlignment(SwingConstants.CENTER);
	lblReproduzir.setFont(new Font("Consolas", Font.PLAIN, 20));
	lblReproduzir.setBounds(208, 158, 122, 24);
	contentPane.add(lblReproduzir);

	JLabel lblVoltarInicio = new JLabel("Voltar Inicio");
	lblVoltarInicio.setFont(new Font("Consolas", Font.PLAIN, 20));
	lblVoltarInicio.setBounds(416, 158, 143, 24);
	contentPane.add(lblVoltarInicio);

	JLabel lblParar = new JLabel("Parar");
	lblParar.setHorizontalAlignment(SwingConstants.CENTER);
```

```
		lblParar.setFont(new Font("Consolas", Font.PLAIN, 20));
		lblParar.setBounds(638, 158, 98, 24);
		contentPane.add(lblParar);

		JLabel lblSair = new JLabel("Sair");
		lblSair.setHorizontalAlignment(SwingConstants.RIGHT);
		lblSair.setFont(new Font("Consolas", Font.PLAIN, 20));
		lblSair.setBounds(560, 346, 98, 24);
		contentPane.add(lblSair);
		setOpacity(0.95f);
		setVisible(true);

		setVisible(true);
	}

	public void setText(String text) {
		LogTextField.setText(text);
	}

	public void setOnOff(boolean b) {
		if(!b) {
			label.setText("Robot Player Desligado");
			label.setForeground(new Color(255, 0, 0));
		}
		else {
			label.setText("Robot Player Ligado");
			label.setForeground(new Color(0, 255, 0));
		}
	}
}
```