

**I**nstituto **S**uperior de **E**ngenharia de **L**isboa

Licenciatura em Engenharia Informática e Multimédia

Semestre de Inverno 2018/2019

Fundamentos de Sistemas Operativos

**Trabalho Prático 2 – TP2**

Trabalho elaborado por:

Fábio Alexandre da Cruz Silva Dias, A42921

Jorge Miguel Coelho Silva, A44615

Docente: Carlos Gonçalves

2018/11/28

Índice

[Indices de Figuras 4](#_Toc532047466)

[1. Introdução 6](#_Toc532047467)

[1.1. O que são *Tarefas* e onde se enquadra neste projeto. 7](#_Toc532047468)

[1.2. Como são implementados estas *Tarefas*? 7](#_Toc532047469)

[1.3. Como se garante “prioridade”? 7](#_Toc532047470)

[1.4. O que é *Exclusão Mútua*? 7](#_Toc532047471)

[1.5. *Semáforos.* 7](#_Toc532047472)

[2. Diagrama de Atividades 8](#_Toc532047473)

[2.1. *Diagrama de Atividades - Geral.* 8](#_Toc532047474)

[2.2. *Diagrama de Atividades – Fugir* 9](#_Toc532047475)

[2.3. *Diagrama de Atividades – Vaguear* 10](#_Toc532047476)

[2.4. *Diagrama de Atividades – Evitar* 11](#_Toc532047477)

[3. Sincronização entre Tarefas 12](#_Toc532047478)

[3.1. *Sincronização no acesso ao recurso partilhado ( robot )* 12](#_Toc532047479)

[3.2. Classe *Comportamentos* 12](#_Toc532047480)

[3.3. Sincronização entre o Fugir e o Vaguear 12](#_Toc532047481)

[3.4. Sincronização entre a tarefa Evitar e as tarefas Fugir e Vaguear 13](#_Toc532047482)

[4. Terminação das tarefas controlado 14](#_Toc532047483)

[5. Diagramas UML de classes 15](#_Toc532047484)

[6. Conclusões 16](#_Toc532047485)

[7. Código do Projeto 17](#_Toc532047486)

[7.1. Classe – *“GUI”* 17](#_Toc532047487)

[7.2. Classe – *“Vaguear”* 25](#_Toc532047488)

[7.3. Classe – *“Evitar”* 26](#_Toc532047489)

[7.4. Classe – *“Fugir”* 27](#_Toc532047490)

[7.5. Classe – *“Comportamentos”* 29](#_Toc532047491)

[7.6. Classe Utils– *“Utils”* 30](#_Toc532047492)

# Indices de Figuras

[Figura 1 - Interface Gráfica (GUI) 6](#_Toc532047493)

[Figura 2 - Diagrama de Atividades (Geral) 8](#_Toc532047494)

[Figura 3 - Diagrama de Actividades - FUGIR 9](file:///S:\ISEL\3º%20Semestre\FSO\TP2\relatorio_Original.docx#_Toc532047495)

[Figura 4 - Diagrama de Actividades - VAGUEAR 10](file:///S:\ISEL\3º%20Semestre\FSO\TP2\relatorio_Original.docx#_Toc532047496)

[Figura 5 - Diagrama de Actividades - EVITAR 11](file:///S:\ISEL\3º%20Semestre\FSO\TP2\relatorio_Original.docx#_Toc532047497)

[Figura 6 - Diagrama UML 15](#_Toc532047498)

# Introdução

Pretende-se o desenvolvimento de uma aplicação para controlo de um robot constituído por 3 **tarefas** **JAVA** que permitam aplicar comportamentos a um robot. Esses comportamentos são então o **Vaguear**, **Evitar Obstáculo** e o **Fugir**.

A comunicação entre estas **tarefas** é suportada através de objetos partilhados e a sincronização utilizando **semáforos** ou **monitores**.

Para a execução deste trabalho, é requisitado que seja modificada/criada uma interface gráfica (**GUI**) com opções, tipo *JCheckBox*[[1]](#footnote-1), para o **Vaguear**, **Evitar** e **Fugir**. Ao ativar esta *CheckBox*, as **tarefas** têm de ser *ativadas* ou *desativadas* de forma independente.

Na próxima figura *(Figura 1)* é apresentada as modificações visuais a nossa Interface Gráfica.

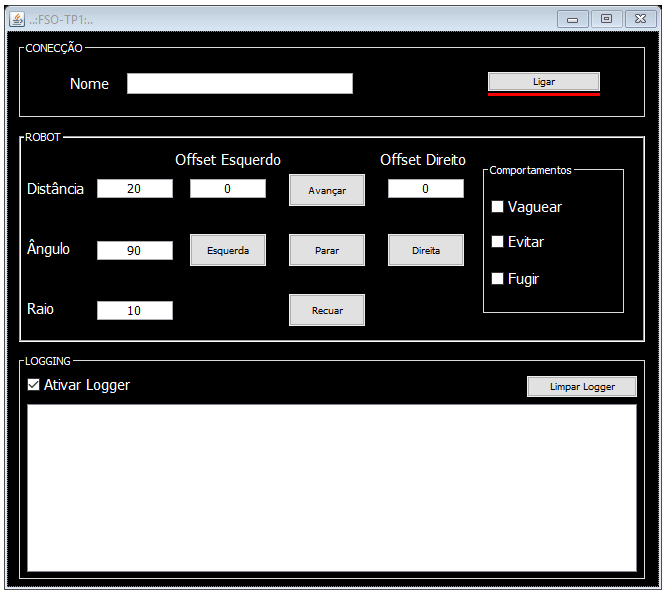


Figura 1 - Interface Gráfica (GUI)

## O que são *Tarefas* e onde se enquadra neste projeto.

*“Uma tarefa é uma atividade ou um trabalho que se têm de executar.”*

Neste projeto é pedido que todos os comportamentos sejam uma tarefa, assim todos os nosso comportamentos terão de seguir algumas “regras” para poderem executar, um dado algoritmo, sem que exista qualquer tipo de interferência com um outro qualquer comportamento.

## Como são implementados estas *Tarefas*?

Cada comportamento que tenha de ser modelado será uma extensão da classe *Thread*[[2]](#footnote-2), isto para, que exista uma certa “prioridade” para cada um dos comportamentos e que se garanta acesso exclusivo ao *recurso*.

## Como se garante “prioridade”?

Para garantir esta “prioridade” é criado um novo conceito de Sincronização entre Tarefas, isto é, cada tarefa vai ser executada e vai obter acesso ao recurso sem que mais nenhuma esteja lá, ou seja, *Exclusão Mútua.*

## O que é *Exclusão Mútua*?

Entende-se por *Exclusão Mútua*, o acesso exclusivo a um *recurso*. Para que isto seja possível, têm de se criar condições que garantam que cada *Tarefa* saiba se pode ou não pode aceder ao *recurso*. Este tipo de acesso pode ser feito com ***Semáforos e/ou Monitores***.

## *Semáforos.*

Um *Semáforo*[[3]](#footnote-3) controla o acesso a um *recurso* que seja partilhado por meio de um contador. Se esse contador for superior a 0, significa que existe acesso a esse recurso, caso não seja superior significa que não existe acesso a esse *recurso*. No caso do contador ser superior a 0, para se obter Exclusão desse semáforo é preciso fazer *semaforo.aquire()*, no qual vai decrementar uma unidade do contador. Quando já não for preciso ter Exclusão é só fazer *semaforo.release()*.

*semaforo.aquire()* – Decrementa uma unidade do contador do semáforo, caso este seja igual a 0, a tarefa ficará em espera, sem gastar recursos.

*semaforo.release() –* Incrementa uma unidade no contador do semáforo.

# Diagrama de Atividades

## *Diagrama de Atividades - Geral.*

No seguinte quadro está um Diagrama de Atividades geral do projeto.

Interface Gráfica

( GUI )

ROBOT

Comportamentos

*java.lang.Thread*

{Estende}

{Estende}

*Vaguear*

*Evitar*

*Fugir*

{Estende}

Figura 2 - Diagrama de Atividades (Geral)

## *Diagrama de Atividades – Fugir*

Figura - Diagrama de Actividades - FUGIR

ROBOT

Comportamentos

{Estende}

*GUI*

*Fugir*

Inicia a sua ação de fugir

Acesso esclusivo?

*false*

{Aguarda acesso}

*true*

Requisitar ao robot

Sensor distancia?

Libertar acesso escliusvo

*Return > 60 cm*

{Não necessita fazer ação}

{aguarda ‘*x’* tempo}

verifica se o evitar está

inactivo e se está livre

*Return < 60 cm*

{Necessita fazer ação}

*GUI*

## *Diagrama de Atividades – Vaguear*

Figura - Diagrama de Actividades - VAGUEAR

Comportamentos

{Estende}

*GUI*

*Vaguear*

Inicia a sua ação vaguear

Acesso esclusivo?

*false*

{Aguarda acesso}

*true*

Libertar acesso escliusvo

{aguarda ‘*x’* tempo}

verifica se o evitar está

inactivo e se está livre

*GUI*

*Robot*

## *Diagrama de Atividades – Evitar*

Figura - Diagrama de Actividades - EVITAR

Comportamentos

{Estende}

*GUI*

*Evitar*

Inicia a sua ação de Evitar

Acesso esclusivo?

*false*

{Aguarda acesso}

*true*

Libertar acesso escliusvo

{aguarda ‘*x’* tempo}

verifica se o evitar está

inactivo e se está livre

*GUI*

*Robot*

# Sincronização entre Tarefas

## *Sincronização no acesso ao recurso partilhado ( robot )*

Como dito anteriormente, para ter acesso ao recurso que irá ser partilhado por todas as tarefas, é necessário criar um acesso de *Exclusão Mutua*, utilizando os *Semáforos*. Para garantirmos esta sincronização foi criada uma classe *Comportamentos* que estende de *Thread*, e todas as outras tarefas, as classes **Vaguear, Fugir e Evitar**, serão uma extensão da classe *Comportamentos*.

Esta classe tem como objetivo declarar funções/métodos obrigatórios para todas a classes que derivam dela. As tarefas que derivam desta classe também têm acesso a dois *Semáforos*, para obter acesso aos recursos que necessitam.

## Classe *Comportamentos*

A classe *Comportamentos*, como dito anteriormente, é uma classe que implementa uma obrigação, funções/métodos, a todas as classes que derivam dela. Uma tarefa ao derivar desta classe, têm a sua disposição, variáveis de controlo e métodos para obter acesso a um recurso ou bloquear outras tarefas de acederem a esse recurso.

Como o recurso do robot só pode ser acedido por cada uma das tarefas em exclusivo, foi necessário, utilizando *Semáforos*, criar exceções de acesso a esse recurso, isto porque, é impossível que duas ou mais tarefas conseguirem enviar comandos ao robot exatamente ao mesmo tempo. Existindo este problema, foi necessário criar dois *semáforos*, um de Controlo (*control*) e outro para aceder ao recurso (*oEngTinhaRazao*).

O *Semáforo* de Controlo (*control*) serve para garantir que se a tarefa pode ou não pode ser executada.

O *Semáforo* de acesso ao recurso (*oEngTinhaRazao*) serve para aceder em exclusão ao recurso, neste caso ao robot.

## Sincronização entre o Fugir e o Vaguear

Vejamos o seguinte exemplo, o robot ao **Vaguear** se algo se aproximar dele, numa distância mínima de 60cm, o robot irá iniciar uma ação de **Fugir** desse objeto.

Mais sucintamente, estas classes, depois de serem inicializadas, têm de obter um qualquer tipo de input para darem *release* ao seu *semáforo* de controlo e ficarem “ativas”. Após o *semáforo* de controlo ter mais que 0 unidades, inicia-se a tentativa de obter acesso exclusivo ao recurso, nesta fase, entra o *semáforo* de acesso ao recurso. Este *semáforo* que é partilhado por todas as tarefas, é inicializado com uma unidade, na *Interface Gráfica* e ao ser feito um *adquire* é garantido que mais nenhuma outra tarefa tenha acesso ao recurso.

## Sincronização entre a tarefa Evitar e as tarefas Fugir e Vaguear

Ao ser inicializada e enviado um input de *“start”*, esta tarefa para iniciar a sua função.

Esta tarefa tem como objetivo requisitar, de 250 em 250ms, o valor do sensor de toque. Este ao responder, quando em contacto “1”, inicia um conjunto de ações.

Estas ações começam por adquirir o *semáforo* de acesso ao robot, bloqueando assim todas as outras classes de acederem ao robot. Após este *semáforo* ficar *“aquired”* esta tarefa envia para o robot um comando para parar logo, recuar 15 cm e curvar a esquerda. Após estas ações serem feitas esta tarefa da *“release”* ao *semáforo* de acesso ao robot, deixando assim, todas as outras tarefas com acesso a aceder ao recurso.

# Terminação das tarefas controlado

Para garantirmos que todas as tarefas terminem como deve de ser, ao enviar-mos um *“input”* para terminarem, todos os *semáforos* de Controlo são postos com 0 unidades fazendo com que as suas ações parem de ser executadas, ficando em modo *idle* sem consumir recursos do CPU.

# Diagramas UML de classes

A close up of text on a white background

Description automatically generated

Figura 6 - Diagrama UML

# Conclusões

A utilização de semáforos, apesar de complicada, facilita muito os trabalhos com “*multi-thread”*, pois é mais fácil “controlar” acessos e ações utilizando estes mêtodos.

Adquirimos bons conhecimentos para a utilização futura de semáforos/monitores e a importância destes agentes na sincronização e organização de acessos a recursos.

# Código do Projeto

## Classe – *“GUI”*

**import** java.awt.Color;

**import** java.awt.ComponentOrientation;

**import** java.awt.Font;

**import** java.awt.event.ActionEvent;

**import** java.awt.event.ActionListener;

**import** java.awt.event.FocusAdapter;

**import** java.awt.event.FocusEvent;

**import** java.awt.event.KeyAdapter;

**import** java.awt.event.KeyEvent;

**import** java.awt.event.MouseAdapter;

**import** java.awt.event.MouseEvent;

**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.JScrollPane;

**import** javax.swing.JTextArea;

**import** javax.swing.JTextField;

**import** javax.swing.SwingConstants;

**import** javax.swing.UIManager;

**import** javax.swing.border.EtchedBorder;

**import** javax.swing.border.TitledBorder;

**public** **class** Gui **extends** Thread {

**private** JFrame frame;

**private** JTextField txtNomeRobot;

**private** JTextField txtDistance;

**private** JTextField txtAngle;

**private** JTextField txtRadius;

**private** JTextField txtOffsetLeft;

**private** JTextField txtOffsetRight;

**private** JTextArea txtrLogging;

**private** JCheckBox chckbxCheckLogging;

**private** JCheckBox chckbxVaguear;

**private** JCheckBox chckbxEvitar;

**private** JCheckBox chckbxFugir;

**private** JButton btnConectar;

**private** JLabel lblConectado;

// Texto pre-definidos

**private** **final** String NOTCONNECTED = "Ligação ao Robot não iniciada...";

// Variaveis Globais

**private** String name;

**private** **int** offSetLeft, offSetRight, angle, distance, radius;

**boolean** robotOn;

Comportamentos vaguear, evitar, fugir;

// myRobot robot;

**private** RobotLegoEV3 robot;

**private** Semaphore oEngTinhaRazao;

**public** **void** run() {

}

/\*\*

\* Launch the application.

\*/

**public** **static** **void** main(String[] args) {

Gui window = **new** Gui();

window.frame.setVisible(**true**);

window.start();

}

/\*\*

\* Iniciar variaveis

\*/

**public** **void** init() {

**this**.name = ""; // default - EV3

**this**.offSetLeft = 0;

**this**.offSetRight = 0;

**this**.angle = 90;

**this**.distance = 20;

**this**.radius = 10;

**this**.robotOn = **false**;

**this**.txtNomeRobot.setText(name);

**this**.txtOffsetLeft.setText(String.*valueOf*(offSetLeft));

**this**.txtOffsetRight.setText(String.*valueOf*(offSetRight));

**this**.txtRadius.setText(String.*valueOf*(radius));

**this**.txtAngle.setText(String.*valueOf*(angle));

**this**.txtDistance.setText(String.*valueOf*(distance));

**this**.robot = **new** RobotLegoEV3();

**this**.oEngTinhaRazao = **new** Semaphore(1);

**this**.vaguear = **new** Vaguear("Vaguear", **this**.oEngTinhaRazao, **this**.robot);

**this**.vaguear.start();

**this**.evitar = **new** Evitar("Evitar", **this**.oEngTinhaRazao, **this**.robot);

**this**.evitar.start();

**this**.fugir = **new** Fugir("Fugir", **this**.oEngTinhaRazao, **this**.robot, **this**.vaguear);

**this**.fugir.start();

}

/\*\*

\* Método para Ligar/Desligar o ROBOT

\*

\* **@throws** InterruptedException

\*/

**private** **void** connectRobot() **throws** InterruptedException {

**if** (robotOn) {

updateConnect(**false**);

robot.CloseEV3();

} **else** {

**if** (name.equals("") || name == **null** || name.length() <= 0) {

txtNomeRobot.setBackground(Color.***RED***);

logger("Nome do Robot vazio...");

} **else** {

logger("A iniciar ligação");

logger("Aguarde...");

**boolean** control = **true**;

**while** (control) {

**if** (robot.OpenEV3(name)) {

updateConnect(**true**);

control = **false**;

} **else** {

**try** {

Thread.*sleep*(250);

} **catch** (InterruptedException e) {

e.printStackTrace();

}

}

}

}

}

}

/\*\*

\*

\* Método para alterar gráficamente a GUI.

\*

\* **@param** value:

\* - true = Altera gráficamente a GUI para indicar que a há

\* Ligação ao Robot. - false = Altera gráficamente a GUI para

\* indicar que não há Ligação ao Robot.

\* **@throws** InterruptedException

\*/

**private** **void** updateConnect(**boolean** value) **throws** InterruptedException {

**if** (value) {

btnConectar.setText("Desligar");

lblConectado.setBackground(Color.***GREEN***);

logger("Ligação ao Robot Concluída com Sucesso!");

robotOn = **true**;

} **else** {

btnConectar.setText("Ligar");

lblConectado.setBackground(Color.***red***);

logger("Ligação ao Robot desligada com sucesso!");

vaguear.Stop();

evitar.Stop();

chckbxEvitar.setSelected(**false**);

chckbxVaguear.setSelected(**false**);

chckbxFugir.setSelected(**false**);

robotOn = **false**;

}

}

/\*\*

\*

\* **@param** backwards:

\* - False = Distnace value takes its integral value - True =

\* Distance value is negated

\*/

**private** **void** move(**boolean** backwards) {

**int** dis = distance;

**if** (robotOn) {

**if** (backwards) {

dis = dis \* -1;

}

robot.Reta(dis);

robot.Parar(**false**);

} **else** {

logger(NOTCONNECTED);

}

}

/\*\*

\* Método para fazer o Robot Virar.

\* **@param** right

\* - True - Virar a direita - False - Virar a Esquerda

\*/

**private** **void** turn(**boolean** right) {

**if** (robotOn) {

**if** (right) {

robot.CurvarDireita(radius, angle);

} **else** {

robot.CurvarEsquerda(radius, angle);

}

robot.Parar(**false**);

} **else** {

logger(NOTCONNECTED);

}

}

/\*\*

\* Método para PARA logo o robot.

\*/

**private** **void** stopMove() {

**if** (robotOn) {

robot.Parar(**true**);

} **else** {

logger(NOTCONNECTED);

}

}

/\*\*

\* Função que regista o texto no logger caso este esteja activo

\*

\* **@param** text

\* - text - Texto para ser inserido no logger (gráfico)

\*/

**public** **void** logger(String text) {

**if** (txtrLogging.isEnabled()) {

txtrLogging.append(text + "\n");

}

}

**public** **void** clearLog() {

**if** (!chckbxCheckLogging.isEnabled()) {

txtrLogging.setEnabled(**true**);

txtrLogging.setText("");

txtrLogging.setEnabled(**false**);

} **else** {

txtrLogging.setText("");

}

}

/\*\*

\* Create the application.

\*/

**public** Gui() {

**super**("GUI");

initialize();

init();

}

/\*\*

\* Initialize the contents of the frame.

\*/

**private** **void** initialize() {

frame = **new** JFrame();

frame.setTitle("..:FSO-TP1:..");

frame.setResizable(**false**);

frame.getContentPane().setBackground(Color.***BLACK***);

frame.setBounds(100, 100, 658, 585);

frame.setDefaultCloseOperation(JFrame.***DISPOSE\_ON\_CLOSE***);

frame.getContentPane().setLayout(**null**);

frame.setLocationRelativeTo(**null**);

JPanel panelConeccao = **new** JPanel();

panelConeccao.setBorder(

**new** TitledBorder(**null**, "CONEC\u00C7\u00C3O", TitledBorder.***LEFT***, TitledBorder.***TOP***, **null**, Color.***WHITE***));

panelConeccao.setBackground(Color.***BLACK***);

panelConeccao.setBounds(10, 10, 630, 79);

frame.getContentPane().add(panelConeccao);

panelConeccao.setLayout(**null**);

JLabel lblNomeRobot = **new** JLabel("Nome");

lblNomeRobot.setHorizontalAlignment(SwingConstants.***LEFT***);

lblNomeRobot.setFont(**new** Font("Tahoma", Font.***PLAIN***, 15));

lblNomeRobot.setForeground(Color.***WHITE***);

lblNomeRobot.setBounds(53, 22, 47, 40);

panelConeccao.add(lblNomeRobot);

txtNomeRobot = **new** JTextField();

txtNomeRobot.addFocusListener(**new** FocusAdapter() {

@Override

**public** **void** focusLost(FocusEvent e) {

name = txtNomeRobot.getText();

**if** (name.length() > 0) {

txtNomeRobot.setBackground(Color.***WHITE***);

}

}

});

txtNomeRobot.addKeyListener(**new** KeyAdapter() {

@Override

**public** **void** keyPressed(KeyEvent e) {

**if** (e.getKeyChar() == KeyEvent.***VK\_ENTER***) {

name = txtNomeRobot.getText();

**if** (name.length() > 0) {

txtNomeRobot.setBackground(Color.***WHITE***);

}

}

}

});

txtNomeRobot.setHorizontalAlignment(SwingConstants.***CENTER***);

txtNomeRobot.setFont(**new** Font("Tahoma", Font.***PLAIN***, 12));

txtNomeRobot.setBounds(110, 32, 226, 21);

panelConeccao.add(txtNomeRobot);

txtNomeRobot.setColumns(10);

btnConectar = **new** JButton("Ligar");

btnConectar.addMouseListener(**new** MouseAdapter() {

@Override

**public** **void** mouseClicked(MouseEvent e) {

**try** {

connectRobot();

} **catch** (InterruptedException e1) {

e1.printStackTrace();

}

}

});

btnConectar.setToolTipText("Ligar Gestor");

btnConectar.setEnabled(**true**);

btnConectar.setFont(**new** Font("Tahoma", Font.***PLAIN***, 10));

btnConectar.setBounds(471, 31, 112, 19);

panelConeccao.add(btnConectar);

lblConectado = **new** JLabel("");

lblConectado.setBackground(Color.***RED***);

lblConectado.setOpaque(**true**);

lblConectado.setForeground(Color.***RED***);

lblConectado.setBounds(471, 52, 112, 3);

panelConeccao.add(lblConectado);

JPanel panelRobot = **new** JPanel();

panelRobot.setBorder(**new** TitledBorder(

**new** EtchedBorder(EtchedBorder.***LOWERED***, **new** Color(255, 255, 255), **new** Color(160, 160, 160)), "ROBOT",

TitledBorder.***LEFT***, TitledBorder.***TOP***, **null**, **new** Color(255, 255, 255)));

panelRobot.setBackground(Color.***BLACK***);

panelRobot.setBounds(10, 99, 630, 214);

frame.getContentPane().add(panelRobot);

panelRobot.setLayout(**null**);

JLabel lblDistancia = **new** JLabel("Dist\u00E2ncia");

lblDistancia.setHorizontalAlignment(SwingConstants.***LEFT***);

lblDistancia.setForeground(Color.***WHITE***);

lblDistancia.setFont(**new** Font("Tahoma", Font.***PLAIN***, 15));

lblDistancia.setBounds(10, 33, 60, 50);

panelRobot.add(lblDistancia);

JLabel lblAngulo = **new** JLabel("\u00C2ngulo");

lblAngulo.setHorizontalAlignment(SwingConstants.***LEFT***);

lblAngulo.setForeground(Color.***WHITE***);

lblAngulo.setFont(**new** Font("Tahoma", Font.***PLAIN***, 15));

lblAngulo.setBounds(10, 93, 60, 50);

panelRobot.add(lblAngulo);

JLabel lblRaio = **new** JLabel("Raio");

lblRaio.setFont(**new** Font("Tahoma", Font.***PLAIN***, 15));

lblRaio.setForeground(Color.***WHITE***);

lblRaio.setBounds(10, 153, 60, 50);

panelRobot.add(lblRaio);

txtDistance = **new** JTextField();

txtDistance.setHorizontalAlignment(SwingConstants.***CENTER***);

txtDistance.setFont(**new** Font("Tahoma", Font.***PLAIN***, 12));

txtDistance.setBounds(80, 49, 76, 19);

panelRobot.add(txtDistance);

txtDistance.setColumns(10);

txtAngle = **new** JTextField();

txtAngle.setFont(**new** Font("Tahoma", Font.***PLAIN***, 12));

txtAngle.setHorizontalAlignment(SwingConstants.***CENTER***);

txtAngle.setBounds(80, 111, 76, 19);

panelRobot.add(txtAngle);

txtAngle.setColumns(10);

txtRadius = **new** JTextField();

txtRadius.setFont(**new** Font("Tahoma", Font.***PLAIN***, 12));

txtRadius.setHorizontalAlignment(SwingConstants.***CENTER***);

txtRadius.setBounds(80, 171, 76, 19);

panelRobot.add(txtRadius);

txtRadius.setColumns(10);

JButton btnAvancar = **new** JButton("Avan\u00E7ar");

btnAvancar.addActionListener(**new** ActionListener() {

**public** **void** actionPerformed(ActionEvent e) {

distance = Integer.*parseInt*(txtDistance.getText());

move(**false**);

}

});

btnAvancar.setFont(**new** Font("Tahoma", Font.***PLAIN***, 10));

btnAvancar.setBounds(272, 44, 76, 32);

panelRobot.add(btnAvancar);

JButton btnParar = **new** JButton("Parar");

btnParar.addMouseListener(**new** MouseAdapter() {

@Override

**public** **void** mouseClicked(MouseEvent e) {

stopMove();

}

});

btnParar.setFont(**new** Font("Tahoma", Font.***PLAIN***, 10));

btnParar.setBounds(272, 104, 76, 32);

panelRobot.add(btnParar);

JButton btnRecuar = **new** JButton("Recuar");

btnRecuar.addMouseListener(**new** MouseAdapter() {

@Override

**public** **void** mouseClicked(MouseEvent e) {

distance = Integer.*parseInt*(txtDistance.getText());

move(**true**);

}

});

btnRecuar.setFont(**new** Font("Tahoma", Font.***PLAIN***, 10));

btnRecuar.setBounds(272, 164, 76, 32);

panelRobot.add(btnRecuar);

JButton btnEsquerda = **new** JButton("Esquerda");

btnEsquerda.addMouseListener(**new** MouseAdapter() {

@Override

**public** **void** mouseClicked(MouseEvent e) {

radius = Integer.*parseInt*(txtRadius.getText());

angle = Integer.*parseInt*(txtAngle.getText());

turn(**false**);

}

});

btnEsquerda.setFont(**new** Font("Tahoma", Font.***PLAIN***, 10));

btnEsquerda.setBounds(173, 104, 76, 32);

panelRobot.add(btnEsquerda);

JButton btnDireita = **new** JButton("Direita");

btnDireita.addMouseListener(**new** MouseAdapter() {

@Override

**public** **void** mouseClicked(MouseEvent e) {

radius = Integer.*parseInt*(txtRadius.getText());

angle = Integer.*parseInt*(txtAngle.getText());

turn(**true**);

}

});

btnDireita.setFont(**new** Font("Tahoma", Font.***PLAIN***, 10));

btnDireita.setBounds(371, 104, 76, 32);

panelRobot.add(btnDireita);

JLabel lblOffsetEsq = **new** JLabel("Offset Esquerdo");

lblOffsetEsq.setVerticalAlignment(SwingConstants.***TOP***);

lblOffsetEsq.setForeground(Color.***WHITE***);

lblOffsetEsq.setFont(**new** Font("Tahoma", Font.***PLAIN***, 15));

lblOffsetEsq.setHorizontalAlignment(SwingConstants.***CENTER***);

lblOffsetEsq.setComponentOrientation(ComponentOrientation.***LEFT\_TO\_RIGHT***);

lblOffsetEsq.setBounds(155, 20, 113, 19);

panelRobot.add(lblOffsetEsq);

txtOffsetLeft = **new** JTextField();

txtOffsetLeft.setHorizontalAlignment(SwingConstants.***CENTER***);

txtOffsetLeft.setFont(**new** Font("Tahoma", Font.***PLAIN***, 12));

txtOffsetLeft.setBounds(173, 49, 76, 19);

panelRobot.add(txtOffsetLeft);

txtOffsetLeft.setColumns(10);

txtOffsetRight = **new** JTextField();

txtOffsetRight.setHorizontalAlignment(SwingConstants.***CENTER***);

txtOffsetRight.setFont(**new** Font("Tahoma", Font.***PLAIN***, 12));

txtOffsetRight.setBounds(371, 49, 76, 19);

panelRobot.add(txtOffsetRight);

txtOffsetRight.setColumns(10);

JLabel lblOffsetDrt = **new** JLabel("Offset Direito");

lblOffsetDrt.setForeground(Color.***WHITE***);

lblOffsetDrt.setFont(**new** Font("Tahoma", Font.***PLAIN***, 15));

lblOffsetDrt.setHorizontalAlignment(SwingConstants.***CENTER***);

lblOffsetDrt.setVerticalAlignment(SwingConstants.***TOP***);

lblOffsetDrt.setBounds(359, 20, 95, 19);

panelRobot.add(lblOffsetDrt);

JPanel panel = **new** JPanel();

panel.setForeground(Color.***WHITE***);

panel.setBackground(Color.***BLACK***);

panel.setBorder(**new** TitledBorder(UIManager.*getBorder*("TitledBorder.border"), "Comportamentos",

TitledBorder.***LEADING***, TitledBorder.***TOP***, **null**, **new** Color(255, 255, 255)));

panel.setBounds(464, 33, 145, 153);

panelRobot.add(panel);

panel.setLayout(**null**);

chckbxEvitar = **new** JCheckBox("Evitar");

chckbxEvitar.addMouseListener(**new** MouseAdapter() {

@Override

**public** **void** mouseClicked(MouseEvent e) {

**if** (chckbxEvitar.isSelected() && robotOn) {

evitar.Start();

} **else** {

**try** {

evitar.Stop();

} **catch** (InterruptedException e1) {

e1.printStackTrace();

}

}

}

});

chckbxEvitar.setBounds(6, 68, 81, 21);

panel.add(chckbxEvitar);

chckbxEvitar.setEnabled(**true**);

chckbxEvitar.setForeground(Color.***WHITE***);

chckbxEvitar.setFont(**new** Font("Tahoma", Font.***PLAIN***, 15));

chckbxEvitar.setBackground(Color.***BLACK***);

chckbxVaguear = **new** JCheckBox("Vaguear");

chckbxVaguear.addMouseListener(**new** MouseAdapter() {

@Override

**public** **void** mouseClicked(MouseEvent e) {

**if** (chckbxVaguear.isSelected() && robotOn) {

vaguear.Start();

vaguear.setIsRunning(**true**);

} **else** {

**try** {

vaguear.Stop();

vaguear.setIsRunning(**false**);

} **catch** (InterruptedException e1) {

e1.printStackTrace();

}

}

}

});

chckbxVaguear.setBounds(6, 33, 81, 21);

panel.add(chckbxVaguear);

chckbxVaguear.setEnabled(**true**);

chckbxVaguear.setFont(**new** Font("Tahoma", Font.***PLAIN***, 15));

chckbxVaguear.setForeground(Color.***WHITE***);

chckbxVaguear.setBackground(Color.***BLACK***);

chckbxFugir = **new** JCheckBox("Fugir");

chckbxFugir.addMouseListener(**new** MouseAdapter() {

@Override

**public** **void** mouseClicked(MouseEvent e) {

**if** (chckbxFugir.isSelected() && robotOn) {

fugir.Start();

} **else** {

**try** {

fugir.Stop();

} **catch** (InterruptedException e1) {

e1.printStackTrace();

}

}

}

});

chckbxFugir.setBounds(6, 105, 81, 21);

panel.add(chckbxFugir);

chckbxFugir.setForeground(Color.***WHITE***);

chckbxFugir.setBackground(Color.***BLACK***);

chckbxFugir.setFont(**new** Font("Tahoma", Font.***PLAIN***, 15));

JPanel panelLogging = **new** JPanel();

panelLogging

.setBorder(**new** TitledBorder(**null**, "LOGGING", TitledBorder.***LEFT***, TitledBorder.***TOP***, **null**, Color.***WHITE***));

panelLogging.setBackground(Color.***BLACK***);

panelLogging.setBounds(10, 323, 630, 228);

frame.getContentPane().add(panelLogging);

panelLogging.setLayout(**null**);

chckbxCheckLogging = **new** JCheckBox("Ativar Logger");

chckbxCheckLogging.addMouseListener(**new** MouseAdapter() {

@Override

**public** **void** mouseClicked(MouseEvent e) {

**if** (chckbxCheckLogging.isSelected()) {

txtrLogging.setEnabled(**true**);

} **else** {

txtrLogging.setEnabled(**false**);

}

}

});

chckbxCheckLogging.setSelected(**true**);

chckbxCheckLogging.setFont(**new** Font("Tahoma", Font.***PLAIN***, 15));

chckbxCheckLogging.setForeground(Color.***WHITE***);

chckbxCheckLogging.setBackground(Color.***BLACK***);

chckbxCheckLogging.setBounds(6, 17, 157, 27);

panelLogging.add(chckbxCheckLogging);

JButton btnClearLogging = **new** JButton("Limpar Logger");

btnClearLogging.addMouseListener(**new** MouseAdapter() {

@Override

**public** **void** mouseClicked(MouseEvent e) {

clearLog();

}

});

btnClearLogging.setFont(**new** Font("Tahoma", Font.***PLAIN***, 10));

btnClearLogging.setBounds(510, 22, 110, 21);

panelLogging.add(btnClearLogging);

JScrollPane spLogging = **new** JScrollPane();

spLogging.setFont(**new** Font("Tahoma", Font.***PLAIN***, 15));

spLogging.setBackground(Color.***BLACK***);

spLogging.setBounds(10, 50, 610, 168);

panelLogging.add(spLogging);

txtrLogging = **new** JTextArea();

txtrLogging.setEditable(**false**);

txtrLogging.setLineWrap(**true**);

spLogging.setViewportView(txtrLogging);

}

}

## Classe – *“Vaguear”*

**import** java.util.concurrent.Semaphore;

**import** Utils.Utils;

**public** **class** Vaguear **extends** Comportamentos {

**private** String nome;

**private** **int** tempo;

**private** **int** MAXDISTANCE = 60;

**private** **int** MAXANGLE = 120;

**private** String[] ACCOES = { "andar", "virar" };

**public** Vaguear(String ThreadName, Semaphore oEngTinhaRazao, RobotLegoEV3 robot) {

**super**(ThreadName, oEngTinhaRazao, robot);

}

**private** **void** randomMove() {

**int** delay = 0;

**try** {

oEngTinhaRazao.acquire();

**int** action = (**int**) (0 + Math.*random*() \* 2);

**int** move, radius, angle, toLeft;

**switch** (ACCOES[action]) {

**case** "andar":

move = (**int**) (1 + Math.*random*() \* MAXDISTANCE);

**int** toBack = (**int**) (0 + Math.*random*() \* 2);

**if** (toBack == 1) {

move = move \* -1;

}

robot.Reta(move);

delay = Utils.*delay*(move, **false**, 0);

**break**;

**case** "virar":

toLeft = (**int**) (0 + Math.*random*() \* 2);

radius = (**int**) (1 + Math.*random*() \* MAXDISTANCE);

angle = (**int**) (1 + Math.*random*() \* MAXANGLE);

**if** (toLeft == 1) {

robot.CurvarEsquerda(radius, angle);

} **else** {

robot.CurvarDireita(radius, angle);

}

delay = Utils.*delay*(radius, **true**, angle);

**break**;

**default**:

**break**;

}

} **catch** (InterruptedException e) {

e.printStackTrace();

} **finally** {

oEngTinhaRazao.release();

**try** {

Thread.*sleep*(delay);

} **catch** (InterruptedException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

}

**public** **void** run() {

**for** (;;) {

**if** (getStatus() == Status.***STOP***) {

**try** {

robot.Parar(**false**);

autoSuspend();

} **catch** (InterruptedException e) {

e.printStackTrace();

}

}

randomMove();

}

}

}

## Classe – *“Evitar”*

**import** java.util.concurrent.Semaphore;

**import** Utils.Utils;

**public** **class** Evitar **extends** Comportamentos {

**public** Evitar(String ThreadName, Semaphore oEngTinhaRazao, RobotLegoEV3 robot) {

**super**(ThreadName, oEngTinhaRazao, robot);

}

**private** **int** checkSensor() {

//try {

// oEngTinhaRazao.acquire();

**return** robot.SensorToque(1);

//} finally {

// oEngTinhaRazao.release();

//}

}

**private** **void** avoid() **throws** InterruptedException {

**int** delay = 0;

**if**(checkSensor() == 1) {

System.***out***.println("EVITAR ON");

**try** {

oEngTinhaRazao.acquire();

robot.SetVelocidade(50);

robot.Parar(**true**);

robot.Reta(-15);

robot.CurvarEsquerda(0, 90);

robot.Parar(**false**);

}**catch** (InterruptedException e) {

e.printStackTrace();

} **finally** {

oEngTinhaRazao.release();

Thread.*sleep*( Utils.*delay*(-15, **false**, 0) + Utils.*delay*(1, **true**, 90));

}

System.***out***.println("EVITAR OFF");

}

}

**public** **void** run() {

**for** (;;) {

**if** (getStatus() == Status.***STOP***) {

**try** {

autoSuspend();

} **catch** (InterruptedException e) {

e.printStackTrace();

}

}

**try** {

avoid();

Thread.*sleep*(250);

} **catch** (InterruptedException e) {

e.printStackTrace();

}

}

}

}

## Classe – *“Fugir”*

**import** java.util.concurrent.Semaphore;

**public** **class** Fugir **extends** Comportamentos {

**private** Comportamentos vaguear;

**private** **enum** runStatus {

***NORMAL***, ***RUN***

};

**private** runStatus status = runStatus.***NORMAL***;

**private** **boolean** memory = **false**;

**private** **final** **int** DELAY = 500;

**private** **final** **int** SAFEDISTANCE = 60;

**private** **int** initialDistance = 0;

**private** **int** actualDistance = 0;

**private** **int** actualSpeed = 50; // default 50%

**public** Fugir(String ThreadName, Semaphore oEngTinhaRazao, RobotLegoEV3 robot, Comportamentos vaguear) {

**super**(ThreadName, oEngTinhaRazao, robot);

**this**.vaguear = vaguear;

}

**private** **void** setNewSpeed(**int** actual, **int** initial) {

**try** {

oEngTinhaRazao.acquire();

**int** delta = actual - initial;

actualSpeed += delta\*-1;

System.***out***.println("Velocidade (antes da censura): " + actualSpeed + " %");

**if**(actualSpeed < 50) {

actualSpeed = 50;

}

**if**(actualSpeed > 100) {

actualSpeed = 100;

}

System.***out***.println("Velocidade (depois da censura): " + actualSpeed + " %");

**int** toRun = actualDistance - SAFEDISTANCE;

robot.Reta(Math.*abs*(toRun));

robot.SetVelocidade(actualSpeed);

} **catch** (InterruptedException e) {

e.printStackTrace();

} **finally** {

oEngTinhaRazao.release();

}

}

**public** **void** run() {

initialDistance = robot.SensorUS(2);

**for** (;;) {

**try** {

Thread.*sleep*(DELAY);

**if** (getStatus() == Status.***STOP***) {

autoSuspend();

}

actualDistance = robot.SensorUS(2);

System.***out***.println("distancia: " + actualDistance);

**if** (vaguear.getIsRunning()) {

memory = **true**;

} **else** {

memory = **false**;

}

**switch** (status) {

**case** ***NORMAL***:

**if** (actualDistance < initialDistance && actualDistance < SAFEDISTANCE) {

status = runStatus.***RUN***;

**if** (memory) {

vaguear.Stop();

}

setNewSpeed(actualDistance, initialDistance);

}

**if** (actualSpeed != 50 && status != runStatus.***RUN***) {

robot.SetVelocidade(50);

}

**break**;

**case** ***RUN***:

**if** (actualDistance < SAFEDISTANCE) {

setNewSpeed(actualDistance, initialDistance);

} **else** {

status = runStatus.***NORMAL***;

setNewSpeed(actualDistance, initialDistance);

robot.Parar(**true**);

**if** (memory) {

vaguear.Start();

memory = **false**;

}

}

**break**;

**default**:

**break**;

}

initialDistance = actualDistance;

} **catch** (InterruptedException e) {

e.printStackTrace();

}

}

}

}

## Classe – *“Comportamentos”*

**import** java.util.concurrent.Semaphore;

**public** **class** Comportamentos **extends** Thread {

**protected** **final** RobotLegoEV3 robot;

**public** **enum** Status {***STOP***, ***RUN***};

**protected** Semaphore control = **new** Semaphore(0);

**protected** **final** Semaphore oEngTinhaRazao;

**protected** Status currentStatus = Status.***STOP***;

**protected** **boolean** isItRunning = **false**;

**public** Comportamentos(String ThreadName, Semaphore oEngTinhaRazao, RobotLegoEV3 robot) {

**super**(ThreadName);

**this**.oEngTinhaRazao = oEngTinhaRazao;

**this**.robot = robot;

}

**public** **boolean** getIsRunning() {

**return** isItRunning;

}

**public** **void** setIsRunning(**boolean** value) {

isItRunning = value;

}

**public** **void** Stop() **throws** InterruptedException {

currentStatus = Status.***STOP***;

}

**public** **void** Start() {

control.release();

}

**public** Status getStatus() {

**return** currentStatus;

}

**public** Semaphore getControl() {

**return** control;

}

**protected** **void** autoSuspend() **throws** InterruptedException {

control.acquire();

currentStatus = Status.***RUN***;

}

}

## Classe Utils– *“Utils”*

**package** Utils;

**public** **class** Utils {

**public** **static** **int** delay(**int** valor, **boolean** raio, **int** angulo) {

// Robot demora 5 segundos a precorrer 100 cm

**if**(valor < 0)

valor = valor \*-1;

**int** convCm = 100;

**int** convMs = 5000;

**int** delay = 0;

**int** aux = valor;

**if** (raio) {

aux = (**int**) (2. \* Math.***PI*** \* valor);

aux = aux \* angulo / 360;

}

delay = aux \* convMs / convCm;

**return** delay;

}

**public** **static** **int** diferential(**int** actualDistance, **int** initialDistance) {

**return** actualDistance - initialDistance;

}

**public** **static** **int** convertionSpeed(**int** distance) {

**float** x0, x1, x2, y0, y2;

**float** y1;

/\*\*

\* x0 = 50cm | y0 = 50%

\* x1 = dist | y1 = x

\* x2 = 0cm | y2 = 75%

\*/

x0 = 50;

x1 = distance;

x2 = 0;

y0 = 50;

y2 = 75;

y1 = ((x1-x0)/(x2-x0))\*(y2-y0)+y0;

**return** (**int**) y1;

}

}

1. - *JCheckBox* – é um componente da biblioteca *java.swing* para interfaces gráficas. Estes componentes têm dois estados, *marcada* (verificada) e *desmarcada* (não verificada). [↑](#footnote-ref-1)
2. - A classe *Thread* é acedida através da biblioteca *java.lang.Thread*. [↑](#footnote-ref-2)
3. - ou *Semaphore* é acedida através da biblioteca *java.util.concurrent.Semaphore* [↑](#footnote-ref-3)