ENGENHARIA INFORMÁTICA E MULTIMÉDIA

2022/2023

FUNDAMENTOS DE SISTEMAS OPERATIVOS

Relatório do Trabalho Prático nº2

10/01/2023

Docente: Carlos Jorge de Sousa Gonçalves

Trabalho realizado por:

António Luís Ferreira, 47500
Tomás Gomes, 48614
Pedro Silva, 48637

Conteúdo

Lista de Figuras

Anexo

1 Introdução

O segundo trabalho prático da Unidade Curricular (UC) de Fundamentos de Sistemas Operativos tem como objetivos a realização de um processo constituido por tarefas em JAVA e a sincronização entre as mesmas utilizando semáforos e monitores. Tambem se pretende desenvolver interfaces gráficas em JAVA Swing utilizando o editor gráfico WindowBuilder.

Pretende-se desenvolver o processo ADMINISTRADOR que é constituido por três tarefas JAVA, a tarefa VAGUEAR, a tarefa EVITAR e a tarefa FUGIR. Ambos o processo e as tarefas conhecem a API do robot.

A interface gráfica do ADMINISTRADOR permite controlar o robot manualmente. Contem três *checkboxes* "Vaguear", "Evitar"e "Fugir", que permitem ativar ou desativar o respetivo comportamento.

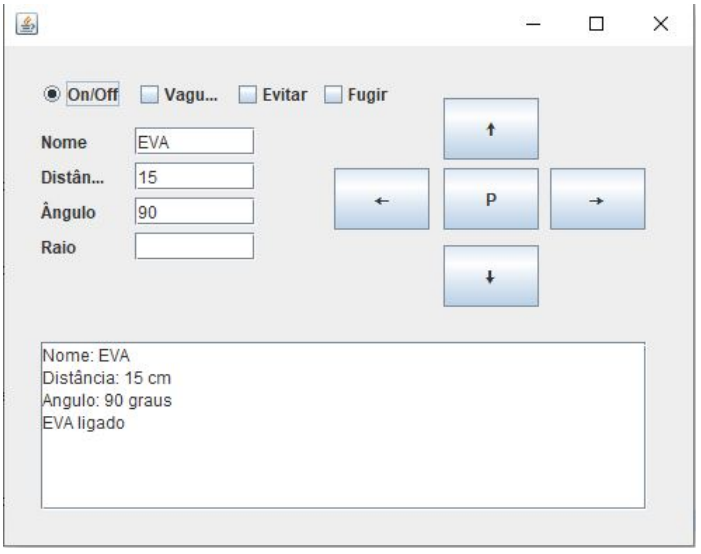

Figura 1: GUI finalizada

O comportamento VAGUEAR é uma tarefa JAVA que faz o robot vaguear pelo espaço. A definição de vaguear no espaço consiste no robot andar em frente, à retaguarda, curvar à direita e curvar à esquerda de forma aleatória. Este comportamento tem uma GUI com uma consola para *debugging*.

O comportamento EVITAR consiste numa tarefa JAVA que quando ativa, repetidamente lê o valor do sensor de toque do robot de 200ms em 200ms e se o valor for 1 então o robot deve parar imediatamente e depois anda 15cm para trás, de seguida faz um curvar à esquerda com raio de 0cm e ângulo de 90 graus e depois pára o robot. Este comportamento deve ter uma GUI com uma consola para *debugging*.

O comportamento FUGIR consiste numa tarefa JAVA que quando ativa, repetidamente lê o valor do sensor de distância do robot de 250ms em 250ms. Se o valor lido for uma

distância inferior a 50cm, então o robot deve andar em frente durante 50 cm e de seguida virar à esquerda ou à direita (escolha aleatória) de 90 graus. O comportamento deve ser executado à velocidade de `80%` e depois regressar à velocidade normal que é de `50%`. Este comportamento deve ter uma GUI com uma consola para *debugging*.

Em termos de funcionalidade simultânea dos três comportamentos, o comportamento EVITAR deve sobrepor-se aos outros dois comportamentos. E o comportamento FUGIR deve sobrepor-se ao comportamento VAGUEAR.

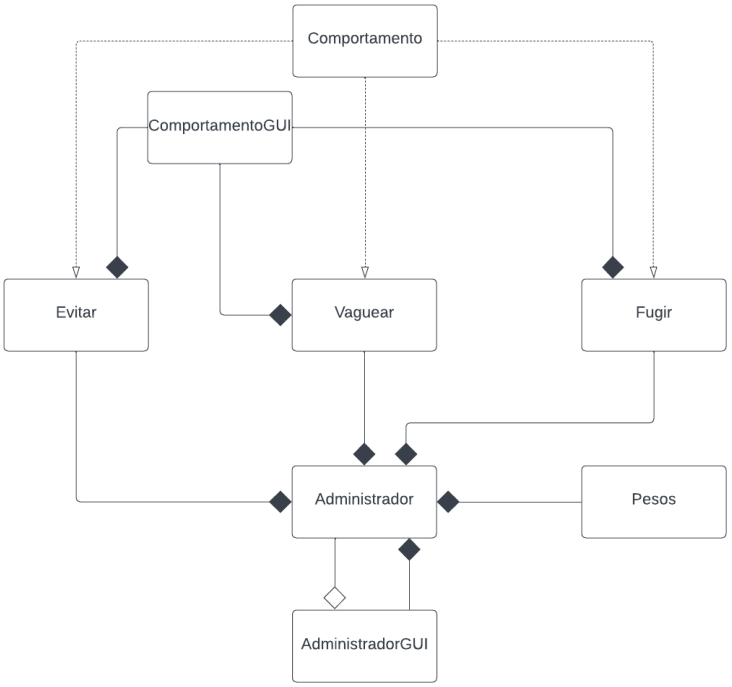

Figura 2: Diagrama de Classes da Aplicação

Como está representado na figura 2, têm-se uma classe principal denominada de Administrador que no seu construtor cria cinco objetos, a sua interface GUI, a classe que controla as prioridades Pesos e os três comportamentos Evitar, Vaguear, Fugir.

Os comportamentos estendem de uma classe abstrata, Comportamento, reune os métodos utilizados pelos comportamentos.

Cada comportamento cria um objeto da classe ComportamentoGUI, classe que instância a interface.

2 Comportamento

A classe Comportamento é uma classe abstrata que estende Thread. Vai servir de base para os outros comportamentos a implementar. Tem quatro estados, `ESPERAR`, `TRABALHAR`, `TERMINAR` e `PERGUNTAR`, e a ComportamentoGUI.

Os comportamentos que derivam desta classe são o Vaguear, Evitar e Fugir

2.1 Atributos

O construtor desta *class* recebe como atríbutos: *Robot*, Pesos e prioridade.

A classe comportamento tem uma referência para a biblioteca do robot, utilizado para sincronizar todos os comportamentos e para o objeto Pesos.

Tem ainda como atributo a prioridade do comportamento para aceder em exclusão ao robot, explicado melhor em Pesos.

2.2 Métodos

- public synchronized void esperar() - Altera o estado do comportamento para esperar;
- public synchronized void trabalha() - Altera o estado do comportamento para trabalhar e notifica o comportamneto;
- public long ExecucaoRetas(int distancia, float vel) - Retorna o tempo de execução de uma reta;
- public long ExecucaoCurvas(int raio, int ang) - Retorna o tempo de execução de uma curva;
- abstract public long gerarMensagem() - Método que será reescrito em canda class que estender de Comportamento;
- abstract public void fecharFrame() - Método que será reescrito em canda class que estender de Comportamento.
- abstract public void gerarMensagem() - Método que gera e envia para o robot um conjunto de mensagens.

3 ComportamentoGUI

A classe ComportamentoGUI é a base da GUI de cada comportamento. Cria um objeto JFrame e um objeto JTextArea onde são escritas as mensagens enviadas para o robot de forma a informar o utilizador e para questões de debuging.

A utilização desta classe pode ser encontrada no subcapítulo do comportamento Vaguear em GUI, para o comportamento Evitar em GUI e para o comportamento Fugir em GUI.

3.1 Atributos

O construtor desta *class* recebe como atributos uma String *título* com o nome do comportamento e um inteiro *local*, com a coordenada x da GUI.

3.2 Métodos

- private void initialize(String titulo, int local) - Inicializa a GUI, cria uma JFrame, uma JTextArea e uma JScrollPane;
- public JTextArea getTextAreaComportamento() - Retorna a textArea.
- public JFrame getFrmComportamento() - Retorna a frame.
- public void dispose() - descarta a frame.

4 Vaguear

A classe Vaguear estende Comportamento. Este comportamento gera instruções aleatórias para o robot realizar como andar em frente, à retaguarda, parar, curvar à direita e à esquerda.

4.1 Construtor

O construtor da *class* faz *super* aos atributos e cria a sua interface gráfica.

4.2 Métodos

- public long gerarMensagem() - Gera uma mensagem aleatória para o robot executar;
- public void run() - Máquina de estados do comportamento.

4.3 GUI

Com recurso à classe ComportamentoGUI o Vaguear cria uma interface no seu contrutor, como podemos observar na figura 3. A GUI do comportamento Vaguear imprime os comandos que são gerados aleatoriamente e enviados para o robot e os seus valores, quer sejam distância, ângulo ou raio.

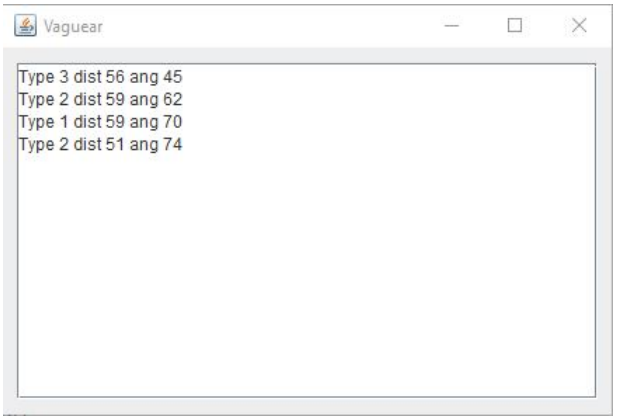

Figura 3: GUI do comportamento Vaguear

4.4 Máquina de Estados

A tarefa Vaguear inicia-se no estado de `ESPERAR`, e muda de estado para `TRABALHAR`, quando a sua respetiva checkBox presente na GUI do Administrador é selecionada. No estado trabalhar, vai ser onde esta tarefa, em exclusão mútua, envia comandos para o robot. Caso a gui do administrador seja fechada, esta passará para o estado `TERMINAR`, onde será interrompida e o programa em geral se fechará.

Como se pode observar na figura 6 do diagrama da respetiva máquina.

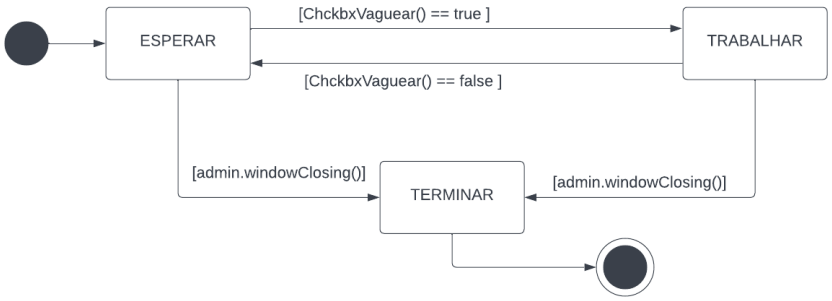

Figura 4: Máquina de estados da tarefa Vaguear

A questão da prioridade entre comportamentos será abordado mais abaixo no subcapitulo Prioridades da classe Pesos.

5 Evitar

A classe Evitar estende Comportamento. Este comportamento pergunta ao robot se o sensor de toque foi ativado a cada 200ms e quando o sensor é ativado gera um conjunto de mensagens para o robot executar.

5.1 Construtor

O construtor da *class* faz *super* aos atributos e cria a sua interface gráfica.

5.2 Métodos

- public long gerarMensagem() - Gera um conjunto de mensagens para o robot executar;
- public void run() - Máquina de estados do comportamento.

5.3 GUI

O comporamento evitar, como já tinha sido referido na introdução, realiza uma sequência de comandos definidos pelo docente, que serão vistos na GUI do comportamento evitar, como podemos observar na figura 5:

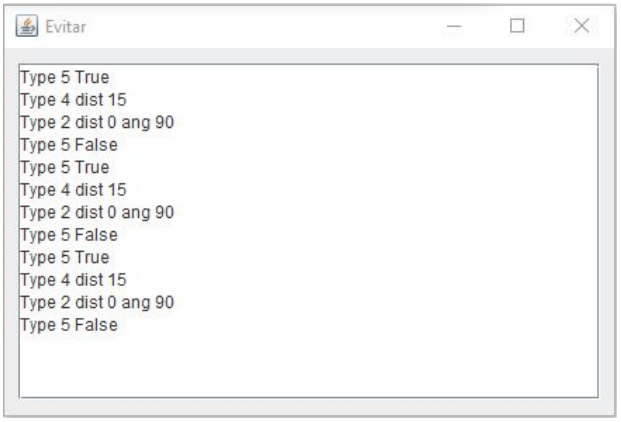

Figura 5: GUI do comportamento Evitar

5.4 Máquina de Estados

A tarefa Evitar inicia-se no estado de `ESPERAR`, e muda de estado para `TRABALHAR`, quando a sua respetiva checkBox presente na GUI do Administrador é selecionada. No estado trabalhar, vai ser onde esta tarefa, em exclusão mútua, pergunta se o sensor de toque está ativo. Caso que sim realiza uma sequência de comandos definidos pelo docente, que serão vistos na GUI. Caso a gui do Administrador seja fechada, esta passará para o estado `TERMINAR`, onde será interrompida e o programa em geral se fechará.

Como se pode observar na figura 6 o diagrama da respetiva máquina.

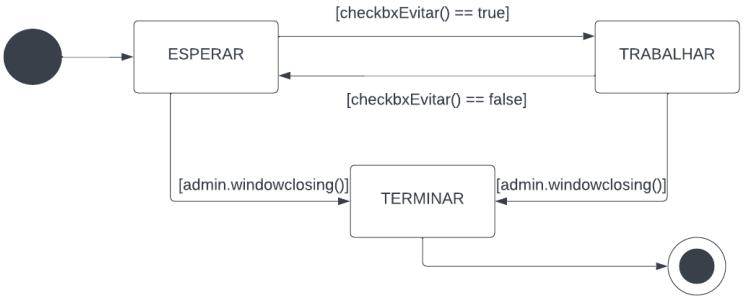

Figura 6: Máquina de estados da tarefa Evitar

A questão da prioridade entre comportamentos será abordado mais abaixo no subcapitulo Prioridades da classe Pesos.

6 Fugir

A classe Fugir estende Comportamento. Este comportamento pergunta ao robot se o sensor de distancia foi ativado a cada 250ms e quando o sensor é ativado gera um conjunto de mensagens para o robot executar.

6.1 Construtor

O construtor da *class* faz *super* aos atributos e cria a sua interface gráfica.

6.2 Métodos

- public long gerarMensagem() - Gera um conjunto de mensagens para o robot executar;
- private int esquerdaOuDiraita() - Méto auxiliar que retorna aleatóriamente esqueda ou diraita;
- public long gerarMensagem() - Gera um conjunto de mensagens para o robot executar;
- public void run() - Máquina de estados do comportamento.

6.3 GUI

Na GUI do comportamento Fugir, podemos observar as mudanças de velocidade e os comandos executados pelo robot, como ilustrado na figura 6:

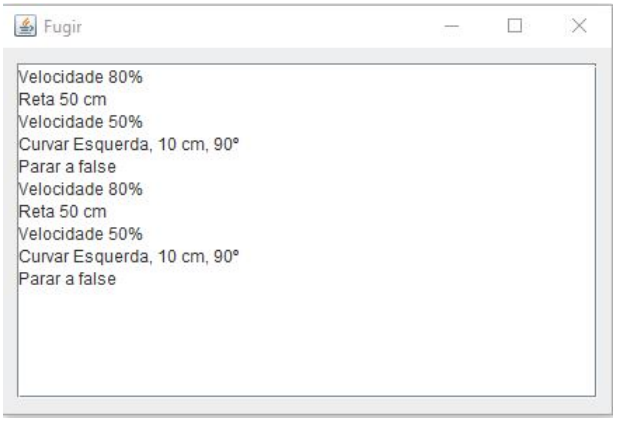

Figura 7: GUI do comportamento Fugir

6.4 Máquina de Estados

A tarefa Fugir inicia-se no estado de `ESPERAR`, e muda de estado para `TRABALHAR`, quando a sua respetiva checkBox presente na GUI do Administrador é selecionada. No estado trabalhar, vai ser onde esta tarefa, em exclusão mútua, pergunta se o sensor de luz está acionado. Caso que sim realiza uma sequência de comandos definidos pelo docente, que serão vistos na GUI. Caso a gui do Administrador seja fechada, esta passará para o estado `TERMINAR`, onde será interrompida e o programa em geral se fechará.

Como se pode observar na figura 8 o diagrama da respetiva máquina.

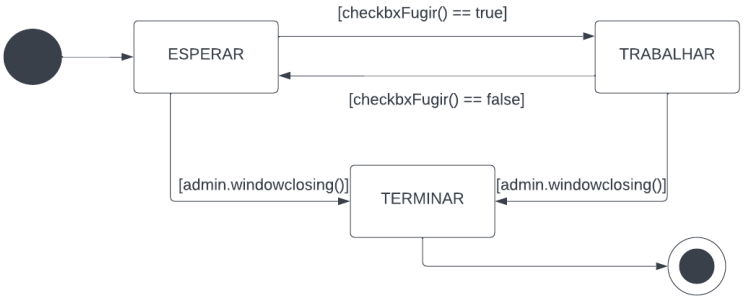

Figura 8: Máquina de estados da tarefa Fugir

A questão da prioridade entre comportamentos será abordado mais abaixo no subcapitulo Prioridades da classe Pesos.

7 Pesos

A classe Pesos é uma classe auxiliar que permite aos comportamentos sobreporem-se uns aos outros, consoantes a sua prioridade definida à *priori*, no construtor da classe Administrador

7.1 Atributos

O único atributo é a variável *prioridadeAtual* que contem o valor da prioridade do último comportamento a adquirir o acesso ao robot, por *default* começa a 0.

7.2 Métodos

- public void verificarPeso(int peso) - verifica se a prioridade do comportamento com acesso ao robot é igual ou mais pequeno que a prioridade do próprio comportamento, caso menor é realizado um *wait*.
- public void resetPeso() - altera o valor da *prioridadeAtual* para 0 e faz um *notifyAll*.

7.3 Prioridades

Para fazer as prioridades (1º Evitar, 2º Fugir, 3º Vaguear), cada comportamento tem a sua prioridade que lhe é atribuído no construtor e sempre que um comportamento acede ao robot, é avaliada a condição no método `verificarPeso()`, se a prioridade for menor que a atual realiza-se um *Thread.wait()*.[1] Em caso contrário a prioridadeAtual passa a ser a prioridade do comportamento que o está a aceder momentaneamente.

Quando cada comportamento está a aceder em exclusão ao *robot* termina a sua ação, este efetua o método resetPeso(), permitindo os comportamentos voltarem a conseguir avaliar a condição de aceder ao *robot*.

8 Administrador

A class Administrador é o "main" da aplicação, onde serão inicializadas as threads e terminadas, consoante as ações do utilizador na GUI.

8.1 Atributos

O administrador tem como atributos a sua GUI, o seu estado, o robot, o peso do comportamento e os três comportamentos. Tem ainda três flags para abrir e fechar os comportamentos, evitando *loops*.

8.2 Métodos

- public void run() - Máquina de estados.
- public void iniciar() - Cria os comportamentos e inicializa as suas threads.
- public void acabar() - Termina os comportamentos e quando estes acabarem também termina.
- public RobotLegoEV3 getRobot() - Retorna a instância do robot.

8.3 GUI

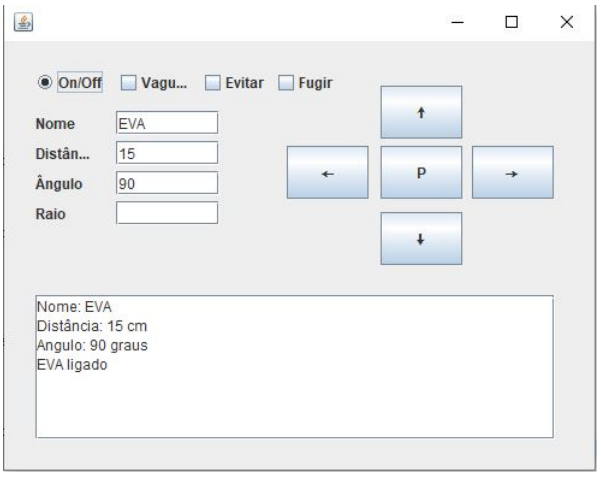

Figura 9: GUI do Administrador

Na figura 9, podemos observar a GUI final do administrador em que iremos controlar o robot por completo. Podemos observar o botão que inicia o robot, o campo que atribui o nome ao robot, os campos para valores, caso queiramos que o robot faça instruções singulares, tais como, por exemplo, uma reta com uma determinada distência, os botões que ativam e desativam cada comportamento, e por fim, uma consola que dá feedback em relação às ações que realizamos na GUI.

Quando é feita a ligação do administrador com o robot os comportamentos são criados e adormecidos. Ao carregar nas checkboxs da GUI o Administrador passa o comportamento selecionado para o estado `TRABALHAR` do comportamento, respetivamente quando se desativa as checkboxs os comportamentos são adormecidos, indo para o estado `ESPERAR` do comportamento. Por fim, quando a interface é encerrada, o método acabar é chamado, terminando os três comportamentos e dando *dispose* das suas respetivas GUIs.

8.4 Máquina de estados

A *Class* Administrador começa no estado `ESPERAR`, onde passa para o estado `TRABALHAR` quando o *radio button* da interface é acionado. No estado `TRABALHAR` é responsável de notificar ou adormecer os comportamentos consoante as *check boxes* dos comportamentos. Caso a janela da interface seja fechada é executado o método **acabar()** e a aplicação é encerrada.

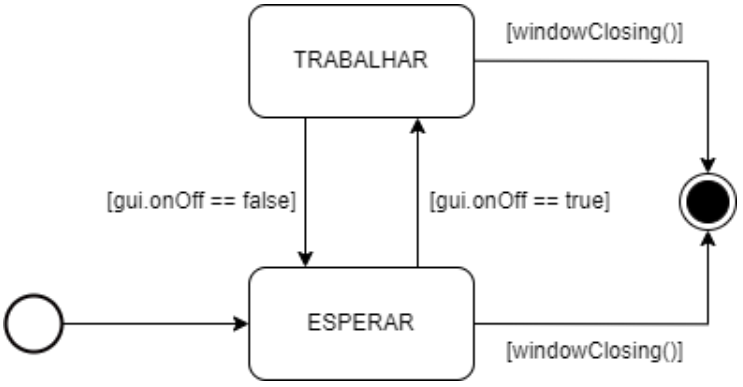

Figura 10: Máquina de estados do administrador

9 Conclusões

Com a realização deste trabalho aprofundamos o nosso conhecimento em relação à matéria[2] lecionada nas aulas, aumentando assim o nosso conhecimento sobre *threads* e sincronismo.

Em suma, foi desenvolvido o processo Administrador que é constituído e controla três tarefas, o Vaguear, o Evitar e o Fugir, que acedem em exclusão mutua a um objeto partilhado, o robot. Os comportamentos estendem uma classe abstrata Comportamento, que por sua vez estende a classe *Thread*. Para garantir o sincronismo entre os vários comportamentos e respeitando as suas Prioridades de acesso ao robot, foram utilizados monitores e a classe auxiliar Pesos.

Desta forma cumprimos os objetivos apresentados para este trabalho.

10 Bibliografia

Referências

[1] Prof. Carlos Gonçalves. *Gestão de Tarefas em Java.* 2019 https://2223moodle.isel.pt/pluginfile.php/1188342/mod_resource/content/2/GestaoDeTarefasEmJava.pdf

[2] Prof. Jorge Pais. *Fundamentos de Sistemas Operativos.* 2022-2023 https://2223moodle.isel.pt/pluginfile.php/1189933/mod_resource/content/3/Fundamentos%20de%20Sistemas%20Operativos%202022-2023%20Vers%C3%A3o%202.pdf

11 Código

```
package package_administrador;

import java.lang.reflect.InvocationTargetException;
import java.util.concurrent.Semaphore;

import package_evitar.Evitar;
import package_fugir.Fugir;
import package_pesos.Pesos;
import package_vaguear.Vaguear;
import robot.RobotLegoEV3;

public class Administrador {
  private AdministradorGUI gui;

  private final int ESPERAR = 0, TRABALHAR = 1;
  private int ESTADO;

  private RobotLegoEV3 robot;
  private Pesos pesos;
  private Vaguear vag;
  private Evitar evi;
  private Fugir fug;

  private boolean vagFlag = false;
  private boolean eviFlag = false;
  private boolean fugFlag = false;

  private final int PRIORIDADE_VAGUEAR = 1;
  private final int PRIORIDADE_EVITAR = 2;
  private final int PRIORIDADE_FUGIR = 3;

  public Administrador() {
    robot = new RobotLegoEV3();
    pesos = new Pesos();
    gui = new AdministradorGUI(this);
    ESTADO = ESPERAR;
    init();
  }

  public void init() {
    vag = new Vaguear(getRobot(), pesos, PRIORIDADE_VAGUEAR);
    evi = new Evitar(getRobot(), pesos, PRIORIDADE_FUGIR);
    fug = new Fugir(getRobot(), pesos, PRIORIDADE_EVITAR);

    vag.start();
    evi.start();
    fug.start();
  }
  public void run() throws InterruptedException,
    InvocationTargetException {
```

```
50
51      for (;;) {
52        switch (ESTADO) {
53        case ESPERAR:
54          if (gui.onOff) {
55            ESTADO = TRABALHAR;
56          }
57          break;
58
59        case TRABALHAR:
60          if (!gui.onOff) {
61            ESTADO = ESPERAR;
62          }
63
64          // Vaguear
65          if (gui.getChckbxVaguear().isSelected() && !vagFlag) {
66            vag.trabalha();
67            vagFlag = true;
68          }
69          if (!gui.getChckbxVaguear().isSelected() && vagFlag) {
70            vag.esperar();
71            vagFlag = false;
72          }
73
74          // Evitar
75          if (gui.getChckbxEvitar().isSelected() && !eviFlag) {
76            evi.trabalha();
77            eviFlag = true;
78          }
79
80          if (!gui.getChckbxEvitar().isSelected() && eviFlag) {
81            evi.esperar();
82            eviFlag = false;
83          }
84
85          // Fugir
86          if (gui.getChckbxFugir().isSelected() && !fugFlag) {
87            fug.trabalha();
88            fugFlag = true;
89
90          }
91
92          if (!gui.getChckbxFugir().isSelected() && fugFlag) {
93            fug.esperar();
94            fugFlag = false;
95          }
96
97          Thread.sleep(1);
98          break;
99
100       }
101     }
```

```
102   }
103
104   public void acabar() throws InterruptedException {
105
106     vag.terminar();
107     vag.getGui().dispose();
108     vag.join();
109
110     evi.terminar();
111     evi.getGui().dispose();
112     evi.join();
113
114     fug.terminar();
115     fug.getGui().dispose();
116     fug.join();
117   }
118
119   public RobotLegoEV3 getRobot() {
120     return robot;
121   }
122
123   // main daqU+FFF[DJ+FFR0D]
124   public static void main(String[] args) throws InterruptedException,
        InvocationTargetException {
125     Administrador re = new Administrador();
126     re.run();
127   }
128
129 }
```

```java
package package_administrador;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JTextField;

import robot.RobotLegoEV3;

import javax.swing.JButton;
import javax.swing.JRadioButton;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import javax.swing.JTextArea;
import javax.swing.JScrollPane;
import javax.swing.JCheckBox;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;

public class AdministradorGUI extends JFrame{

  private JFrame frame;
  private JTextField textNome;
  private JTextField textDist;
  private JTextField textAng;
  private JTextField textRaio;
  private JTextArea textArea;

  protected JCheckBox chckbxVaguear;
  protected JCheckBox chckbxEvitar;
  protected JCheckBox chckbxFugir;

  private String nome;
  private int dist;
  private int raio;
  private int angulo;

  protected boolean onOff;
  private Administrador admin;

  private RobotLegoEV3 robot;

  //construtor da GUI
  public AdministradorGUI() {
    initialize();
  }

  public boolean isOnOff() {
      return onOff;
  }

  public void setOnOff(boolean onOff) {
      this.onOff = onOff;
  }
```

```
53
54  public void setAdministrador(Administrador admin) {
55    this.admin = admin;
56  }
57
58  public AdministradorGUI(Administrador admin) {
59      this();
60      this.setAdministrador(admin);
61  }
62
63
64  /**
65   * Initialize the contents of the frame.
66   */
67  private void initialize() {
68    frame = new JFrame();
69    frame.addWindowListener(new WindowAdapter() {
70
71      @Override
72      public void windowClosing(WindowEvent arg0) {
73        try {
74          admin.acabar();
75        } catch (InterruptedException e) {
76          e.printStackTrace();
77        }
78        System.out.println("Robot desligado...");
79
80        if(robot != null) {
81          robot.Parar(true);
82          try {
83            Thread.sleep(250);
84          } catch (InterruptedException e1) {
85            e1.printStackTrace();
86          }
87
88          robot.CloseEV3();
89        }
90        else {
91          System.exit(0);
92        }
93
94      }
95    });
96    frame.setBounds(100, 100, 507, 395);
97    //frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
98    frame.getContentPane().setLayout(null);
99
100   JScrollPane scrollPane = new JScrollPane();
101   scrollPane.setBounds(26, 211, 432, 120);
102   frame.getContentPane().add(scrollPane);
103
104   textArea = new JTextArea();
```

```
105     scrollPane.setViewportView(textArea);
106
107
108     JLabel lblName = new JLabel("Nome");
109     lblName.setBounds(26, 61, 46, 14);
110     frame.getContentPane().add(lblName);
111
112     JLabel lblDist = new JLabel("Dist[U+FFFD]ncia");
113     lblDist.setBounds(26, 86, 46, 14);
114     frame.getContentPane().add(lblDist);
115
116     JLabel lblAng = new JLabel("[U+FFFD]ngulo");
117     lblAng.setBounds(26, 111, 46, 14);
118     frame.getContentPane().add(lblAng);
119
120     JLabel lblRaio = new JLabel("Raio");
121     lblRaio.setBounds(26, 136, 46, 14);
122     frame.getContentPane().add(lblRaio);
123
124
125
126     textNome = new JTextField();
127     textNome.addActionListener(new ActionListener() {
128       //definir nome
129       public void actionPerformed(ActionEvent e) {
130         nome = textNome.getText();
131         textArea.append("Nome: " + nome + "\n");
132       }
133     });
134     textNome.setBounds(93, 58, 86, 20);
135     frame.getContentPane().add(textNome);
136     textNome.setColumns(10);
137
138     textDist = new JTextField();
139     textDist.addActionListener(new ActionListener() {
140       //definir dist
141       public void actionPerformed(ActionEvent e) {
142       dist = Integer.parseInt(textDist.getText());
143       textArea.append("Dist[U+FFFD]ncia: " + dist + " cm\n");
144       }
145     });
146     textDist.setBounds(93, 83, 86, 20);
147     textDist.setColumns(10);
148     frame.getContentPane().add(textDist);
149
150     textAng = new JTextField();
151     textAng.addActionListener(new ActionListener() {
152       public void actionPerformed(ActionEvent e) {
153         angulo = Integer.parseInt(textAng.getText());
154         textArea.append("Angulo: " + angulo + " graus\n");
155       }
156     });
```

```
157     textAng.setBounds(93, 108, 86, 20);
158     textAng.setColumns(10);
159     frame.getContentPane().add(textAng);
160 
161     textRaio = new JTextField();
162     textRaio.addActionListener(new ActionListener() {
163       public void actionPerformed(ActionEvent e) {
164         raio = Integer.parseInt(textRaio.getText());
165         textArea.append("Raio: " + raio + " cm\n");
166       }
167     });
168     textRaio.setBounds(93, 133, 86, 20);
169     textRaio.setColumns(10);
170     frame.getContentPane().add(textRaio);
171 
172     JButton btnFrente = new JButton("\u25B2");
173     btnFrente.addActionListener(new ActionListener() {
174       public void actionPerformed(ActionEvent arg0) {
175         textArea.append("Reta de: " + dist + " cm\n");
176         robot.Reta(dist);
177         robot.Parar(false);
178       }
179     });
180     btnFrente.setBounds(313, 37, 68, 44);
181     frame.getContentPane().add(btnFrente);
182 
183     JButton btnStop = new JButton("P");
184     btnStop.addActionListener(new ActionListener() {
185       public void actionPerformed(ActionEvent e) {
186 
187         textArea.append("O Robot parou\n");
188         robot.Parar(true);
189 
190       }
191     });
192     btnStop.setBounds(313, 87, 68, 44);
193     frame.getContentPane().add(btnStop);
194 
195     JButton btnRetaguarda = new JButton("\u25BC");
196     btnRetaguarda.addActionListener(new ActionListener() {
197       public void actionPerformed(ActionEvent e) {
198         textArea.append("Retaguarda de: " + (-dist) + " cm\n");
199         robot.Reta(-dist);
200         robot.Parar(false);
201       }
202     });
203     btnRetaguarda.setBounds(313, 142, 68, 44);
204     frame.getContentPane().add(btnRetaguarda);
205 
206     JButton btnDir = new JButton("\u25B6");
207     btnDir.addActionListener(new ActionListener() {
208       //curvar direita
```

```
209       public void actionPerformed(ActionEvent e) {
210         textArea.append("Curva direita com raio de " + raio + " cm e
      " + angulo + " graus\n");
211         robot.CurvarDireita(raio, angulo);
212         robot.Parar(false);
213       }
214     });
215     btnDir.setBounds(389, 87, 68, 44);
216     frame.getContentPane().add(btnDir);
217
218     JButton btnEsq = new JButton("");
219     btnEsq.addActionListener(new ActionListener() {
220       //curvar esquerda
221       public void actionPerformed(ActionEvent e) {
222
223         textArea.append("Curva esquerda com raio de " + raio + " cm
    e " + angulo + " graus\n");
224         robot.CurvarEsquerda(raio, angulo);
225         robot.Parar(false);
226       }
227     });
228     btnEsq.setBounds(235, 87, 68, 44);
229     frame.getContentPane().add(btnEsq);
230
231     JRadioButton rdbtnOnOff = new JRadioButton("On/Off");
232     rdbtnOnOff.addActionListener(new ActionListener() {
233
234       public void actionPerformed(ActionEvent e) {
235         onOff =! onOff;
236           if(onOff) {
237             textArea.append(nome + " ligado\n");
238             robot = admin.getRobot();
239             System.out.println(robot);
240             robot.OpenEV3(nome);
241             onOff= true;
242           }
243           else {
244             textArea.append(nome + " desligado\n");
245               robot.CloseEV3();
246               onOff = false;
247           }
248       }
249     });
250     rdbtnOnOff.setBounds(24, 23, 68, 23);
251     frame.getContentPane().add(rdbtnOnOff);
252
253     //Vaguear
254     chckbxVaguear = new JCheckBox("Vaguear");
255     chckbxVaguear.setBounds(93, 23, 68, 23);
256     frame.getContentPane().add(chckbxVaguear);
257
258     //Evitar
```

```
259     chckbxEvitar = new JCheckBox("Evitar");
260     chckbxEvitar.setBounds(163, 23, 59, 23);
261     frame.getContentPane().add(chckbxEvitar);
262 
263     //Fugir
264     chckbxFugir = new JCheckBox("Fugir");
265     chckbxFugir.setBounds(224, 23, 59, 23);
266     frame.getContentPane().add(chckbxFugir);
267 
268     frame.setVisible(true);
269   }
270 
271   public JTextArea getTextArea() {
272     return textArea;
273   }
274 
275   public JCheckBox getChckbxVaguear() {
276     return chckbxVaguear;
277   }
278 
279   public JCheckBox getChckbxEvitar() {
280     return chckbxEvitar;
281   }
282 
283   public JCheckBox getChckbxFugir() {
284     return chckbxFugir;
285   }
286 }
```

```java
package package_comportamento;

import java.awt.event.WindowEvent;
import java.util.concurrent.Semaphore;

import package_pesos.Pesos;
import robot.RobotLegoEV3;

public abstract class Comportamento extends Thread {

  protected RobotLegoEV3 robot;
  private ComportamentoGUI gui;

  protected static final int ESPERAR = 0;
  protected static final int TRABALHAR = 1;
  protected static final int TERMINAR = 2;

  protected int state;
  protected Pesos pesos;
  protected int prioridade;

  public Comportamento(RobotLegoEV3 robot,Pesos pesos, int prioridade)
    {
    this.state = ESPERAR;
    this.robot = robot;
    this.prioridade = prioridade;
    this.pesos = pesos;
    robot.SetVelocidade(50);
  }

  public synchronized void esperar() {
    this.state = ESPERAR;
  }

  public synchronized void trabalha() {
    this.state = TRABALHAR;
    this.notify();
  }

  public synchronized void terminar() {
    this.state = TERMINAR;
    this.interrupt();

  }

  // (Vrobot = 30cm/s)
  // Tempo de espera retas
  public long ExecucaoRetas(int distancia, float vel) {
    return (long) ((distancia / vel) * 1500.0f);
  }

  // Tempo de espera curvas
```

```
52   public long ExecucaoCurvas(int raio, int ang) {
53     return (long) ((((ang / 360.0f) * 2 * Math.PI * raio) / 30.0f) *
       1500.0f);
54   }
55
56   abstract public long gerarMensagem();
57 }
```

```java
package package_comportamento;

import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;

import javax.swing.JFrame;
import javax.swing.JTextArea;
import javax.swing.JScrollPane;

public class ComportamentoGUI {

  private JFrame frmComportamento;

  private JTextArea textAreaComportamento;

  public ComportamentoGUI(String titulo, int local) {
    initialize(titulo, local);
  }

  /**
   * Initialize the contents of the frame.
   */
  private void initialize(String titulo, int local) {

    frmComportamento = new JFrame();
    frmComportamento.setTitle(titulo);
    frmComportamento.setBounds(local, 100, 450, 300);
    frmComportamento.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
    frmComportamento.getContentPane().setLayout(null);

    JScrollPane scrollPane = new JScrollPane();
    scrollPane.setBounds(10, 11, 414, 239);
    frmComportamento.getContentPane().add(scrollPane);

    textAreaComportamento = new JTextArea();
    scrollPane.setViewportView(textAreaComportamento);
    frmComportamento.setVisible(true);
  }

  public JTextArea getTextAreaComportamento() {
    return textAreaComportamento;
  }

  public JFrame getFrmComportamento() {
    return frmComportamento;
  }

  public void dispose() {
    frmComportamento.setVisible(false);
    frmComportamento.dispose();
  }

```

```
53 }
```

```java
package package_evitar;

import java.awt.event.WindowEvent;
import java.util.concurrent.Semaphore;

import package_comportamento.Comportamento;
import package_comportamento.ComportamentoGUI;
import package_pesos.Pesos;
import robot.RobotLegoEV3;

public class Evitar extends Comportamento{

  private ComportamentoGUI gui;
  private int local = 1050;
  private String msg;

  public Evitar(RobotLegoEV3 robot, Pesos pesos, int prioridade) {
    super(robot,pesos, prioridade);
    gui = new ComportamentoGUI("Evitar",local);
  }

  public ComportamentoGUI getGui() {
    return gui;
  }

  @Override
  public long gerarMensagem() {

    float vel = 30.0f; // cm/s
    int dist = 15;
    int execCurva = 2500; // tempo que demora a executar a m[U+FFFD][U+FFFD]

    robot.Parar(true);
    msg = "Parar a true";
    gui.getTextAreaComportamento().append(msg + "\n");

    robot.Reta(-15);
    msg = "Retaguarda" + " dist " + dist;;
    gui.getTextAreaComportamento().append(msg + "\n");

    robot.CurvarEsquerda(0, 90);
    msg = "Curva para a Esquerda" + " dist " + dist + " ang 90";
    gui.getTextAreaComportamento().append(msg + "\n");

    robot.Parar(false);
    msg = "Parar a true";
    gui.getTextAreaComportamento().append(msg + "\n");

    long tempoEspera = ExecucaoRetas(dist, vel) + execCurva;
    return tempoEspera;
  }

```

```
53  /**
54   *
55   */
56  public void run() {
57    long tempoEspera = 0;
58    while (state != TERMINAR) {
59
60      switch (this.state) {
61
62      case ESPERAR:
63        try {
64          synchronized (this) {
65            this.wait();
66          }
67        } catch (InterruptedException e) {
68          e.printStackTrace();
69        }
70        break;
71
72      case TRABALHAR:
73        try {
74          pesos.verificarPeso(prioridade);
75
76          if (robot.SensorToque(robot.S_1) == 1) {
77            if (state != ESPERAR) {
78              tempoEspera = gerarMensagem();
79              pesos.resetPeso();
80              Thread.sleep(tempoEspera);
81            }
82          }
83          pesos.resetPeso();
84
85        } catch (InterruptedException e1) {
86          e1.printStackTrace();
87          if (state != TERMINAR)
88            esperar();
89        }
90
91        try {
92          Thread.sleep(250);
93        } catch (InterruptedException e) {
94          e.printStackTrace();
95        }
96        break;
97      }
98
99    }
100   robot.Parar(true);
101 }
102
103
104
```

```
105 }
```

```java
package package_fugir;

import java.awt.event.WindowEvent;
import java.util.Random;
import java.util.concurrent.Semaphore;

import package_comportamento.Comportamento;
import package_comportamento.ComportamentoGUI;
import package_pesos.Pesos;
import robot.RobotLegoEV3;

public class Fugir extends Comportamento {

  private ComportamentoGUI gui;
  private int local = 1490;
  private String msg;

  public Fugir(RobotLegoEV3 robot,Pesos pesos, int prioridade) {
    super(robot, pesos, prioridade);
    gui = new ComportamentoGUI("Fugir", local);
  }

  public ComportamentoGUI getGui() {
    return gui;
  }

  @Override
  public long gerarMensagem() {

    float vel = 30.0f; // cm por s
    int dist = 50;
    int radius = 10;
    int ang = 90;

    robot.Parar(true);

    robot.SetVelocidade(80);
    msg = "Velocidade 80%";
    gui.getTextAreaComportamento().append(msg + "\n");

    robot.Reta(50);
    msg = "Reta 50 cm";
    gui.getTextAreaComportamento().append(msg + "\n");

    robot.SetVelocidade(50);
    msg = "Velocidade 50%";
    gui.getTextAreaComportamento().append(msg + "\n");

    if (esquerdaOuDireita() > 0.5) {
      robot.CurvarEsquerda(10, 90);
      msg = "Curvar Esquerda, 10 cm,[90FF];
      gui.getTextAreaComportamento().append(msg + "\n");
```

```
53      }
54
55      else {
56        robot.CurvarDireita(10, 90);
57        msg = "Curvar Direita, 10 cm,[90FFF]";
58      }
59
60      robot.Parar(false);
61      msg = "Parar a false";
62      gui.getTextAreaComportamento().append(msg + "\n");
63
64      long tempoEspera = ExecucaoRetas(dist, vel) + ExecucaoCurvas(radius
        , ang);
65      return tempoEspera;
66    }
67
68    private int esquerdaOuDireita() {
69      Random r = new Random();
70      return r.nextInt(2);
71    }
72
73    public void run() {
74      long tempoEspera = 0;
75      while (state != TERMINAR) {
76
77        switch (this.state) {
78
79        case ESPERAR:
80          try {
81            synchronized (this) {
82              this.wait();
83            }
84          } catch (InterruptedException e) {
85            e.printStackTrace();
86          }
87          break;
88
89        case TRABALHAR:
90          try {
91            pesos.verificarPeso(prioridade);
92
93            if (robot.SensorLuz(robot.S_2) < 500) {
94              if (state != ESPERAR) {
95                tempoEspera = gerarMensagem();
96                pesos.resetPeso();
97                Thread.sleep(tempoEspera);
98              }
99            }
100           pesos.resetPeso();
101
102         } catch (InterruptedException e2) {
103           // TODO Auto-generated catch block
```

```
104             e2.printStackTrace();
105             if (state != TERMINAR)
106               esperar();
107           }
108
109           try {
110             Thread.sleep(250);
111           } catch (InterruptedException e1) {
112             e1.printStackTrace();
113
114           }
115           break;
116
117         }
118       }
119       robot.Parar(true);
120     }
121 }
```

```java
package package_vaguear;

import java.util.Random;
import java.util.concurrent.Semaphore;

import package_comportamento.Comportamento;
import package_comportamento.ComportamentoGUI;
import package_pesos.Pesos;
import robot.RobotLegoEV3;

public class Vaguear extends Comportamento {

  private ComportamentoGUI gui;

  public Vaguear(RobotLegoEV3 robot,Pesos pesos, int prioridade) {
    super(robot, pesos, prioridade);
    gui = new ComportamentoGUI("Vaguear",600);
  }

  public ComportamentoGUI getGui() {
    return gui;
  }

  @Override
  public long gerarMensagem() {

    Random rnd = new Random();

    int[] types = new int[] { 1, 2, 3, 4, 5 };
    int type = 0;
    int lastType = 0;
    float vel = 30.0f; // cm/s

    long tempoEspera = 0;

    while (type == lastType) {
      type = types[rnd.nextInt(5)];
    }

    lastType = type;
    int dist = 50 + rnd.nextInt(10);
    int angulo = 45 + rnd.nextInt(45);
    String msg = null;
    switch (type) {

    case 1:
      msg = "Reta" + " dist " + dist;
      gui.getTextAreaComportamento().append(msg + "\n");
      robot.Reta(dist);
      tempoEspera = ExecucaoRetas(dist, vel);
      break;

```

```
53      case 2:
54        msg = "Curva para a Esquerda" + " dist " + dist + " ang " +
      angulo;
55        gui.getTextAreaComportamento().append(msg + "\n");
56        robot.CurvarEsquerda(dist, angulo);
57        tempoEspera = ExecucaoCurvas(dist, angulo);
58        break;
59
60      case 3:
61        msg = "Curva para a Direita" + " dist " + dist + " ang " + angulo
      ;
62        gui.getTextAreaComportamento().append(msg + "\n");
63        robot.CurvarDireita(dist, angulo);
64        tempoEspera = ExecucaoCurvas(dist, angulo);
65        break;
66
67      case 4:
68        msg = "Retaguarda" + " dist " + dist;
69        gui.getTextAreaComportamento().append(msg + "\n");
70        robot.Reta(-dist);
71        tempoEspera = ExecucaoRetas(dist, vel);
72        break;
73
74      case 5:
75        msg = "Parar a true";
76        gui.getTextAreaComportamento().append(msg + "\n");
77        robot.Parar(true);
78        break;
79      }
80      return tempoEspera;
81    }
82
83    public void run() {
84      long tempoEspera = 0;
85
86      while (state != TERMINAR) {
87
88        switch (this.state) {
89
90        case ESPERAR:
91          try {
92            pesos.verificarPeso(prioridade);
93            robot.Parar(false);
94            pesos.resetPeso();
95
96            synchronized (this) {
97              this.wait();
98            }
99          } catch (InterruptedException e) {
100           e.printStackTrace();
101         }
102
```

```
103         break;
104
105       case TRABALHAR:
106         try {
107           pesos.verificarPeso(prioridade);
108           tempoEspera = gerarMensagem();
109           pesos.resetPeso();
110           Thread.sleep(tempoEspera);
111
112         } catch (InterruptedException e) {
113           e.printStackTrace();
114
115           if (state != TERMINAR)
116             esperar();
117         }
118         break;
119       }
120     }
121
122     robot.Parar(true);
123   }
124
125 }
```

```java
package package_pesos;

public class Pesos {

  private int pesoNoSemaforo = 0;

  public Pesos() {}

  public synchronized void verificarPeso(int peso) throws
    InterruptedException {
    while (peso < pesoNoSemaforo) {
      this.wait();
    }
    pesoNoSemaforo = peso;

  }

  public synchronized void resetPeso() {
    pesoNoSemaforo = 0;
    this.notifyAll();
  }
}
```