ADEETC

Licenciatura em Engenharia

Informática e Multimédia

1º Semestre 16/17 SI

Fundamentos de Sistemas Operativos

Relatório do Trabalho nº 2

Eng. Carlos Gonçalves

Grupo 11

Trabalho realizado por:

Rui Santos nº39286

Diogo Fernandes nº39205

Hugo Safara nº40614

**Índice**

[**Introdução** 2](#_Toc468481879)

[**Desenvolvimento** 3](#_Toc468481880)

[**Vaguear** 3](#_Toc468481881)

[**Evitar Obstáculo** 8](#_Toc468481882)

[**Gestor** 11](#_Toc468481883)

[**Robot Interface** 13](#_Toc468481884)

[**MailBox** 16](#_Toc468481885)

[**Conclusão** 18](#_Toc468481886)

[**Bibliografia** 19](#_Toc468481887)

# **Introdução**

Este trabalho tem como objetivo, fazer com que o robot vagueie pelo seu meio de forma aleatória e que quando encontra um obstáculo, consiga evita-lo. Com base nas aulas teóricas, foi-se aprofundando e compreendo melhor o que era necessário fazer para que este trabalho corresse bem. Foi nos introduzido as ”caixas de correio”, que é uma classe responsável por criar um ficheiro de texto *.dat* que irá servir de ponto de comunicação entre os diversos processos, pois os processos não podem comunicar entre si. Neste caso as caixas de correio serão optadas por um ficheiro de texto mas, poderia não ser de tal maneira. Este trabalho é então a criação dos processos vaguear, evitar obstáculo e o gestor.

A implementação do comportamento VAGUEAR consiste num processo JAVA que quando ativo, abre o canal de comunicação Bluetooth com o robot, configura o sensor de toque no robot e depois faz com que o robot vagueie pelo espaço até encontrar um obstáculo. Quando o robot encontra um obstáculo, o sensor de toque do robot que normalmente está desativo fica ativo e neste caso, o processo deve parar o robot, fechar o canal de comunicação e ficar desativo. A definição de vaguear no espaço consiste no robot andar em frente, curvar à direita, curvar à esquerda e parar de forma aleatória.

A implementação do comportamento EVITAR OBSTÁCULO consiste num processo JAVA que quando ativo, abre o canal de comunicação Bluetooth com o robot, configura o sensor de toque no robot e se a leitura do sensor de toque devolver 1 então o robot anda 15cm para trás e depois faz um curvar à esquerda com raio de 0cm e ângulo de 90 graus e depois para o robot. Se o sensor de toque continuar ativo deve repetir o procedimento anterior senão deve fechar o canal de comunicação Bluetooth e ficar desativo.

A implementação do comportamento GESTOR consiste num processo JAVA que comunica bidireccionalmente com os processos VAGUEAR e EVITAR OBSTÁCULO. A comunicação entre processos permite ao GESTOR ativar e desativar qualquer um dos outros processos, tal como os outros processos quando ficam desativos devem informar o GESTOR através do canal de comunicação.

# **Desenvolvimento**

## **Vaguear**

Neste processo, o robot irá circular de forma “aleatória” no meio onde se encontra. As direções que ele tomará, dependerão das várias instruções que o “vaguear” gera para o robot. Basicamente, o processo gera direções aleatórias e o robot irá mover-se de acordo com as direções que recebeu, parando, quando encontra um obstáculo. As várias direções que ele pode tomar são: frente, curvar à direita, curvar à esquerda e parar. O processo principal “vaguear” implementa *ILogger*. É através do *ILogger* que se poderá, utilizando depois o método *public String log()*, imprimir na consola da interface gráfica algo para fins de *debug*.

public class Vaguear **implements** ILogger **{**

public final int MinForward **=** 50**;** // Centimeters

public final int RndForward **=** 40**;** // Centimeters

public final int MinRadius **=** 50**;** // Centimeters

public final int RndRadius **=** 30**;** // Centimeters

public final int MinAngle **=** 30**;** // Degrees

public final int RndAngle **=** 60**;** // Degrees

public final int SensorSleepTime **=** 250**;** // Milliseconds

public enum Directions **{**

Stop**,** Right**,** Left**,** Forward

**};**

public Directions currentDirection**;**

private MailBox mailVaguear**;**

private MailBox mailGestor**;**

private String prefixGestor **=** "PG"**;**

private String prefixVaguear **=** "PV"**;**

private String prefixGUI **=** "PGUI"**;**

protected String robotName**;**

protected Random rnd**;**

protected MyRobot theRobot**;**

private boolean guiOrGestor**;**

Nesta figura, tem-se inicialização e declaração das variáveis que irão ser utilizadas nos métodos para assim conseguir implementar corretamente o “vaguear”. Como se pode verificar no inicio do programa, é criado um enumerado (enum) que contém as direções possíveis que o robot poderá optar. Foram criadas as caixas de correio, para se conseguir comunicar com um gestor de processos e com o próprio processo em questão. É ainda aqui, que se inicializa o robot, um *boolean* que irá decidir se o processo será corrido num robot ou apenas em teste e outras inicializações importantes para a construção dos métodos.

O construtor Vaguear, recebe como parâmetros o nome do robot, o sensor de toque do robot que irá permitir saber quando o processo Vaguear termina e dois *booleans* para simular o robot em termos físicos ou de testes, e ainda verificar se a ordem é da GUI ou do GESTOR. É ainda dentro do construtor que se instancia duas *MailBox* (ou caixas de correio) para o gestor e para o processo vaguear, a velocidade dos motores e a inicialização do sensor de toque.

public Vaguear**(**String name**,** int touchSensor**,** boolean simulateRobot**,** boolean guiOrGestor**)** **{**

mailVaguear **=** **new** MailBox**(**"vaguear.dat"**);**

mailGestor **=** **new** MailBox**(**"gestor.dat"**);**

**this.**guiOrGestor **=** guiOrGestor**;**

**this.**robotName **=** name**;**

**this.**currentDirection **=** Directions**.**Stop**;**

**this.**rnd **=** **new** Random**();**

**this.**theRobot **=** **new** MyRobot**(**simulateRobot**,** **this);**

**if** **(this.**theRobot**.**OpenNXT**(this.**robotName**)** **==** **false)** **{**

String message**;**

message **=** **this.**log**(**"Could not connect to robot %s"**,** **this.**robotName**);**

**throw** **new** IllegalArgumentException**(**message**);**

**}**

**this.**theRobot**.**SetSpeed**(**50**);**

**this.**theRobot**.**SetTouchSensor**(**touchSensor**);**

**}**

private Directions getNextDirection**()** **{**

int aux**;**

aux **=** **this.**rnd**.**nextInt**(**90**);**

**if** **(**aux **<** 30**)** **{**

**return** Directions**.**Right**;**

**}**

**if** **(**aux **<** 60**)** **{**

**return** Directions**.**Left**;**

**}**

**return** Directions**.**Forward**;**

**}**

private int getRandomRadius**()** **{**

**return** **this.**MinRadius **+** **this.**rnd**.**nextInt**(this.**RndRadius**);**

**}**

private int getRandomAngle**()** **{**

**return** **this.**MinAngle **+** **this.**rnd**.**nextInt**(this.**RndAngle**);**

**}**

private int getRandomDistance**()** **{**

**return** **this.**MinForward **+** **this.**rnd**.**nextInt**(this.**RndForward**);**

**}**

private double getCurveDistance**(**double angle**,** double radius**)** **{**

double perimeter **=** 2.0 **\*** Math**.**PI **\*** radius**;**

**return** angle **\*** perimeter **/** 360.0**;**

**}**

O método privado *getNextDirection()*, trata de retornar uma direção de acordo com um valor inteiro que é gerado. Se o valor inteiro for menor que 30, o método retorna a direção “curvar à direita” para o robot mais tarde cumprir. Se o valor inteiro estiver entre 30 e 60, retornará “curvar à esquerda”, caso o inteiro não esteja dentro destas condições, o método retorna a direção para ir em frente. Os métodos *getRandomRadius()*, *getRandomAngle()* e *getRandomDistance()* tratam de gerar valores aleatórios de raio, ângulo e distâncias respetivamente. O método *getCurveDistance()* retorna um *double* com a distancia que o robot fez aquando da direção que lhe foi gerada.

public void readMailBox**()** **{**

**while** **(!**mailVaguear**.**read**().**startsWith**(**prefixGestor**))** **{**

**try** **{**

Thread**.**sleep**(**500**);**

**}** **catch** **(**InterruptedException e**)** **{**

e**.**printStackTrace**();**

**}**

**}**

**try** **{**

**this.**log**(**"verifica content PVAGUEAR:------------------- " **+** mailVaguear**.**read**());**

**if** **(**mailVaguear**.**read**().**contains**(**"start"**))** **{**

**this.**mailVaguear**.**eraseContent**();**

doWork**();**

**}**

// else if

**}** **catch** **(**Exception e**)** **{**

// TODO Auto-generated catch block

e**.**printStackTrace**();**

**}**

**}**

É neste método readMailBox(), que o processo vaguear irá saber se será processado ou não, isto tudo através do mailBox() que é criado por um gestor. Este gestor é quem dita quem irá ser executado e quando. Os processos que ele administra são o de vaguear e do evitar obstáculo somente. Se ao ler-se a caixa de correio do vaguear não contiver o prefixo do gestor, o processo tem de ficar em espera ate próxima instrução. Caso contenha esse prefixo, o “vaguear” tem luz verde para executar tudo.

Isto permite que quando o gestor lançe o processo vaguear, ele fique continuamente à espera de ler algo na sua caixa de correio que o permita começar o seu comportamento.

public void doWork**()** **throws** Exception **{**

boolean work**;**

work **=** **true;**

double radius**,** angle**,** distance**;**

double sleepTime **=** 0.0**;**

**for** **(;** work **==** **true;)** **{**

Directions newDirection**;**

**while** **((**newDirection **=** getNextDirection**())** **==** **this.**currentDirection**)**

**;**

**this.**currentDirection **=** newDirection**;**

**switch** **(this.**currentDirection**)** **{**

**case** Right**:**

radius **=** getRandomRadius**();**

angle **=** getRandomAngle**();**

sleepTime **=** getSleepTime**(**radius**,** angle**);**

**this.**theRobot**.**CurvarDireita**((**int**)** radius**,** **(**int**)** angle**);**

**this.**log**(**"Right(%3.2f, %3.2f)->%3.2f"**,** radius**,** angle**,** getCurveDistance**(**radius**,** angle**));**

**break;**

**case** Forward**:**

distance **=** getRandomDistance**();**

sleepTime **=** getSleepTime**(**distance**);**

**this.**theRobot**.**Reta**((**int**)** distance**);**

**this.**log**(**"Forward(%3.2f)"**,** distance**);**

**break;**

**case** Left**:**

radius **=** getRandomRadius**();**

angle **=** getRandomAngle**();**

sleepTime **=** getSleepTime**(**radius**,** angle**);**

**this.**theRobot**.**CurvarEsquerda**((**int**)** radius**,** **(**int**)** angle**);**

**this.**log**(**"Left(%3.2f, %3.2f)->%3.2f"**,** radius**,** angle**,** getCurveDistance**(**radius**,** angle**));**

**break;**

**default:**

**break;**

**}**

// Uncomment next line to force a stop after each movement

// this.theRobot.Parar( false );

**this.**log**(**"Sleep(%3.2f)"**,** sleepTime**);**

É neste método *doWork()*, que o robot irá, efetivamente, vaguear. Dentro do ciclo *for*, é instanciado um novo objeto Directions que verifica se a direção atual é igual à seguinte. Caso for, o robot cumpre essa direção de acordo com o *sleepTime* existente em cada *case* do *switch*. Caso não for, cumpre uma nova direção e assim sucessivamente até encontrar um obstáculo e parar de vez. Para cada direção, é gerado um valor aleatório de raio, de angulo e de tempo de execução da direção.

Ainda dentro do *doWork(),* dependendo se o utilizador estiver a trabalhar com a GUI ou apenas com o gestor, o *if* so será corrido se o boolean guiOrGestor for *false*, se for *false*, estar-se-á a trabalhar com o gestor se for *true*, salta para o *else* e executa as instruções para a GUI. Basicamente, no if, caso tenha sido detetado um obstáculo, o sensor de toque fica ativo e o vaguear escreve no maiGestor a mensagem “PV sstop”, que indica que o processo “vaguear” tem de terminar e o robot parar. No else, o vaguear vai ler da sua própria caixa de correio, se contiver o prefixo da Gui e o “para”, o processo do vaguear deve terminar e o robot parar.

**if** **(!this.**guiOrGestor**)** **{**

**if** **(this.**waitForDistanceAndTestSensor**((**int**)** **(**sleepTime **\*** 1000.0**))** **==** **true)** **{**

**this.**theRobot**.**Parar**(true);**

**this.**log**(**"Colision"**);**

**this.**mailGestor**.**write**(**prefixVaguear **+** "sstop"**);**

work **=** **false;**

**this.**theRobot**.**CloseNXT**();**

readMailBox**();**

**}**

**}**

**else** **{**

Thread**.**sleep**((**int**)(**sleepTime**\***1000.0**));**

**if** **(**mailVaguear**.**read**().**contains**(**prefixGUI **+** "para"**))** **{**

**this.**mailVaguear**.**eraseContent**();**

**this.**mailVaguear**.**closeChannel**();**

**this.**theRobot**.**Parar**(true);**

**this.**theRobot**.**CloseNXT**();**

work **=** **false;**

**}**

**}**

## **Evitar Obstáculo**

public AvoidObstacle**(**String name**,** int sensor**,** boolean simulateRobot**,** boolean guiorgestor**)** **{**

**this.**robotName **=** name**;**

**this.**guiorgestor **=** guiorgestor**;**

**this.**simulate **=** simulateRobot**;**

**this.**sensor **=** sensor**;**

// Cada classe lê o seu proprio mail mas escreve no mail do destinatário

**this.**mailEvitar **=** **new** MailBox**(**"evitar.dat"**);**

**this.**mailGestor **=** **new** MailBox**(**"gestor.dat"**);**

**if** **(!this.**guiorgestor**)** **{**

**this.**theRobot **=** **new** MyRobot**(**simulateRobot**,** **this);**

**if** **(this.**theRobot**.**OpenNXT**(this.**robotName**)** **==** **false)** **{**

String message**;**

message **=** **this.**log**(**"Could not connect to robot %s"**,** **this.**robotName**);**

**throw** **new** IllegalArgumentException**(**message**);**

**}**

**this.**theRobot**.**SetSpeed**(**50**);**

**this.**theRobot**.**SetTouchSensor**(**sensor**);**

**}**

**}**

O construtor do *AvoidObstacle* tem uma particularidade diferente do *Vaguear*, quando queremos que o robot faça o comportamento evitar obstáculo ao comando da GUI, não podemos realizar uma conexão ao robot assim que lançamos o processo, pois se já estamos em comunicação com o robot assim que ligamos a GUI iria dar exceção ao tentar-mos criar uma nova ligação, contra isso foi adicionada uma condição de verificação se o *AvoidObstacle* está a ser lançado pela *guiorgestor.*

public void doAvoidObstacle**()** **{**

**while** **(**readSensor**())** **{**

**try** **{**

**this.**theRobot**.**Reta**(-**MAX\_DISTANCE**);**

**this.**theRobot**.**CurvarEsquerda**(**MAX\_RADIUS**,** MAX\_ANGLE**);**

Thread**.**sleep**(**600**);**

**}** **catch** **(**InterruptedException e**)** **{**

// TODO Auto-generated catch block

e**.**printStackTrace**();**

**}**

**this.**theRobot**.**Parar**(true);**

**}**

**this.**theRobot**.**CloseNXT**();**

**if** **(!this.**guiorgestor**)** **{**

**this.**mailGestor**.**write**(**prefixAvoid **+** "sstop"**);**

**}**

readMailBox**();**

**}**

Este método é o que fará ao robot, efetivamente, evitar o obstáculo. Este método executa sempre o mesmo comportamento ao contrário do vaguear. Quando o sensor de toque é detetado e o processo do vaguear é terminado, o processo evitar obstáculo é posto em marcha e o que ele faz, ao robot, é apenas andar para trás e virar para a esquerda de acordo com certos valores que são passados em MAX\_RADIUS, MAX\_ANGLE e MAX\_DISTANCE e de seguida para. Caso o sensor não esteja ativo, o evitar obstáculo irá escrever na GUI ou no gestor “PA sstop” e aí pode-se, novamente, dar inicio ao processo vaguear. É neste método que a caixa de correio do evitar obstáculo, está constantemente a ler da sua caixa de correio para verificar o que fazer e o que executar.

public void readMailBox**()** **{**

**while** **(!**mailEvitar**.**read**().**startsWith**(**prefixGestor**))** **{**

**try** **{**

Thread**.**sleep**(**600**);**

**}** **catch** **(**InterruptedException e**)** **{**

e**.**printStackTrace**();**

**}**

**}**

**try** **{**

**this.**log**(**"verifica content PEVITAR:------------------- " **+** **this.**mailEvitar**.**read**());**

**if(**mailEvitar**.**read**().**contains**(**"PGUIstart"**)){**

**this.**log**(**"ENTREI NO PGUIStart"**);**

mailEvitar**.**eraseContent**();**

**this.**theRobot **=** **new** MyRobot**(**simulate**,** **this);**

**if** **(this.**theRobot**.**OpenNXT**(this.**robotName**)** **==** **false)** **{**

String message**;**

message **=** **this.**log**(**"Could not connect to robot %s"**,**

**this.**robotName**);**

**throw** **new** IllegalArgumentException**(**message**);**

**}**

**this.**theRobot**.**SetSpeed**(**50**);**

**this.**theRobot**.**SetTouchSensor**(**sensor**);**

doAvoidObstacle**();**

**}**

**else** **if** **(**mailEvitar**.**read**().**contains**(**"start"**))** **{**

// tem de consumir

**this.**mailEvitar**.**eraseContent**();**

doAvoidObstacle**();**

**}** **else** **if** **(**mailEvitar**.**read**().**contains**(**"sstop"**))** **{**

**this.**mailEvitar**.**eraseContent**();**

**this.**mailEvitar**.**closeChannel**();**

**this.**theRobot**.**Parar**(true);**

**this.**theRobot**.**CloseNXT**();**

**}**

**}** **catch** **(**Exception e**)** **{**

// TODO Auto-generated catch block

e**.**printStackTrace**();**

**}**

**}**

Esta é a caixa de correio do evitar obstáculo. Se ao ler-se a caixa de correio do evitar obstáculo não contiver o perfixo do gestor, o processo tem de ficar em espera ate próxima instrução. Só quando ele lê da sua própria caixa de correio e verifica se contém la a mensagem “PGUI Start”, é que criada uma ligação ao robot. Dessa forma, fica possível executar o *doAvoidObstacle()* através da GUI. Caso contenha a mensagem “start”, que é a mensagem vinda do gestor, o processo inicia o *doAvoidObstacle().* E caso contenha “sstop”, o processo fica a saber que é para parar o robot e o processo em questão.

## **Gestor**

private static final int SLEEP\_TIME **=** 1000**;**

private MailBox mailGestor**,** mailVaguear**,** mailEvitar**;**

private String jarAvoid **=** "C:\\Users\\Denga\\Desktop\\JARFILES\\evitar.jar"**;**

private String jarVaguear **=** "C:\\Users\\Denga\\Desktop\\JARFILES\\vaguear.jar"**;**

private Process processVaguear**,** processAvoid**;**

private String**[]** arguments **=** **new** String**[]{**"java"**,**"-jar"**,**"GUIA4"**,**""**+**RobotLego**.**S\_2**,**"true"**};**

private String prefixGestor **=** "PG"**;**

private String prefixAvoid **=** "PA"**;**

private String prefixVaguear **=** "PV"**;**

private String prefixGUI **=** "PGUI"**;**

private int estado**;**

static private final int init **=** 0**;**

static private final int vaguear **=** 1**;**

static private final int avoid **=** 2**;**

static private final int Terminar **=** 5**;**

public Gestor**()** **{**

**this.**mailGestor **=** **new** MailBox**(**"gestor.dat"**);**

**this.**mailEvitar **=** **new** MailBox**(**"evitar.dat"**);**

**this.**mailVaguear **=** **new** MailBox**(**"vaguear.dat"**);**

**this.**estado **=** init**;**

**}**

No processo gestor, é criada uma conexão entre todas as caixas de correio existentes, a do *gestor, vaguear* e *evitar* *obstáculo*. São também escritos os caminhos dos jar’s do evitar e do *vaguear* para assim ser possível lançar os processos dentro do gestor. São também criados os estados para o autómato que se irá seguir. O construtor *Gestor()* apenas instancia MailBox’s dos vários processos e que o estado inicial do autómato é igual a *init*.

public void launchProcesses**()** **{**

**while** **(**estado **!=** Terminar**)** **{**

**switch** **(**estado**)** **{**

**case** init**:**

- Estado incial do gestor.

- Escreve na caixa de correio vaguear uma permissão;

- Adição dos argumentos em List<String> respetivos ao vaguear e ao evitar obstáculo, para o lançamento do processo;

- Estado seguinte é igual ao do vaguear (estado = vaguear).

**case** vaguear**:**

**-** Gera um pequeno delay para garantir o lançamento do vaguear;

- O gestor lê da sua própria caixa de correio e se receber a mensagem “PG para” sabe que todos os processos têm de terminar (estado=Terminar);

- Se a caixa de correiodo gestor contiver a mensagem “PV sstop”, o processo evitar obstáculo é posto em ação;

- Caso não cumpra nenhuma das anteriores, permanece no seu próprio estado.

**case** avoid**:**

**-** Gera um pequeno delay para garantir o lançamento do evitar;

- O gestor Lê da sua própria caixa de correio e se receber a mensagem “PG para” sabe que todos os processos têm de terminar (estado=Terminar);

- Se a caixa de correio do gestor contiver a mensagem “PA sstop”, o processo vaguear é posto em ação;

- Caso não cumpra nenhuma das anteriores, permanece no seu próprio estado.

C:\Users\Hugo Safara\Downloads\Untitled Diagram.png

## **Robot Interface**

À classe RobotInterface, já criada do trabalho nº1, adicionou-se:

private MailBox mailGUI**,** mailVaguear**,** mailEvitar**,** mailGestor**;**

private String**[]** arguments **=** **new** String**[]** **{** "java"**,** "-jar"**,** "GUIA4"**,** "" **+** RobotLego**.**S\_2**,** "true" **};**

private String jarGestor **=** "C:\\Users\\Denga\\Desktop\\JARFILES\\gestor.jar"**;**

private String jarVaguear **=** "C:\\Users\\Denga\\Desktop\\JARFILES\\vaguear.jar"**;**

private String jarEvitar **=** "C:\\Users\\Denga\\Desktop\\JARFILES\\evitar.jar"**;**

private Process processoGestor**,**processoVaguear**,**processoEvitar**;**

private String prefixGUI **=** "PGUI"**,** prefixVaguear **=** "PV"**,** prefixEvitar **=** "PA"**,** prefixGestor **=** "PG"**;**

É através destas variáveis que será possível correr, na interface gráfica, os processos vaguear, evitar obstáculo e Gestor. Simplesmente, copiou-se a inicialização das variáveis no gestor e adicionou-se à classe RobotInterface.

chckbxHandler **=** **new** JCheckBox**(**"Handler"**);**

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

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

gestor **=** **!**gestor**;**

chckbxEvitar**.**setEnabled**(!**gestor**);**

chckbxWander**.**setEnabled**(!**gestor**);**

launchHandler**(**gestor**);**

**}**

**});**

chckbxHandler**.**setBounds**(**30**,** 283**,** 97**,** 23**);**

contentPane**.**add**(**chckbxHandler**);**

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

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

public void actionPerformed**(**ActionEvent arg0**)** **{**

evitar **=** **!**evitar**;**

chckbxHandler**.**setEnabled**(!**evitar**);**

chckbxWander**.**setEnabled**(!**evitar**);**

launchAvoid**(**evitar**);**

**}**

**});**

chckbxEvitar**.**setBounds**(**30**,** 257**,** 97**,** 23**);**

contentPane**.**add**(**chckbxEvitar**);**

chckbxWander **=** **new** JCheckBox**(**"Wander"**);**

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

public void actionPerformed**(**ActionEvent arg0**)** **{**

vaguear **=** **!**vaguear**;**

chckbxEvitar**.**setEnabled**(!**vaguear**);**

chckbxHandler**.**setEnabled**(!**vaguear**);**

launchVaguear**(**vaguear**);**

**}**

**});**

chckbxWander**.**setBounds**(**30**,** 233**,** 89**,** 23**);**

contentPane**.**add**(**chckbxWander**);**

private void launchHandler(boolean condition) {

if (condition) {

try {

List<String> argsHandler;

argsHandler = new ArrayList<String>();

argsHandler.add(arguments[0]);

argsHandler.add(arguments[1]);

argsHandler.add(jarGestor);

// caminho do jar vaguear

ProcessBuilder pbGestor;

pbGestor = new ProcessBuilder(argsHandler);

pbGestor.redirectErrorStream(true);

pbGestor.inheritIO();

this.processoGestor = pbGestor.start();

} catch (IOException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}else{

try {

this.mailGestor.write(prefixGUI+ "para");

//espera que o processo acabe as suas tarefas e so depois mata o processo

this.processoGestor.waitFor();

} catch (InterruptedException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

}

Criaram-se *checkboxes* para se conseguir correr os processos vaguear, evitar e gestor na interface gráfica.

A cada “checkbox” que criamos agregadamente criamos também um método por cada lançamento de processo, *vaguear, evitar obstáculo* ou *gestor*.

private void launchHandler**(**boolean condition**)** **{**

**if** **(**condition**)** **{**

**try** **{**

List**<**String**>** argsHandler**;**

argsHandler **=** **new** ArrayList**<**String**>();**

argsHandler**.**add**(**arguments**[**0**]);**

argsHandler**.**add**(**arguments**[**1**]);**

argsHandler**.**add**(**jarGestor**);**

// caminho do jar vaguear

ProcessBuilder pbGestor**;**

pbGestor **=** **new** ProcessBuilder**(**argsHandler**);**

pbGestor**.**redirectErrorStream**(true);**

pbGestor**.**inheritIO**();**

**this.**processoGestor **=** pbGestor**.**start**();**

**}** **catch** **(**IOException e**)** **{**

// TODO Auto-generated catch block

e**.**printStackTrace**();**

**}**

**}else{**

**try** **{**

**this.**mailGestor**.**write**(**prefixGUI**+** "para"**);**

//espera que o processo acabe as suas tarefas e so depois mata o processo

**this.**processoGestor**.**waitFor**();**

**}** **catch** **(**InterruptedException e**)** **{**

// TODO Auto-generated catch block

e**.**printStackTrace**();**

**}**

**}**

**}**

Como em todo o resto da nossa interface gráfica, temos variáveis booleanas que representam o estado de uma checkbox, se está selecionada ou não, respetivamente *true* or *false*, são essas mesmas que quando alteradas na acção da checkbox, são passadas como parâmetro no método de lançamento, indicando assim que quando a variável está a *false*, significa que queremos encerrar o processo, escrevendo na caixa de correio do gestor “PGUIpara” e este quando lê encarrega-se de fechar os sub-processos.

Para este processo apenas precisamos dos argumentos *java -jar caminhodojar* visto o Gestor não ter parâmetros de entrada relevantes para o seu funcionamento.

private void launchVaguear**(**boolean condition**){**

**if** **(**condition**)** **{**

**try** **{**

**this.**mailVaguear**.**write**(**prefixGestor **+** "start"**);**

List**<**String**>** argsWander**;**

argsWander **=** **new** ArrayList**<**String**>();**

argsWander**.**add**(**arguments**[**0**]);**

argsWander**.**add**(**arguments**[**1**]);**

argsWander**.**add**(**jarVaguear**);**

argsWander**.**add**(**arguments**[**2**]);**

argsWander**.**add**(**arguments**[**3**]);**

argsWander**.**add**(**arguments**[**4**]);**

//indica se 頡penas a GUI a iniciar ou Gestor

//true - GUI

//false - Gestor

argsWander**.**add**(**"true"**);**

// caminho do jar vaguear

ProcessBuilder pbVaguear**;**

pbVaguear **=** **new** ProcessBuilder**(**argsWander**);**

pbVaguear**.**redirectErrorStream**(true);**

pbVaguear**.**inheritIO**();**

**this.**processoVaguear **=** pbVaguear**.**start**();**

**}** **catch** **(**IOException e**)** **{**

// TODO Auto-generated catch block

e**.**printStackTrace**();**

**}**

**}else{**

**try** **{**

**this.**mailVaguear**.**write**(**prefixGUI**+** "para"**);**

//espera que o processo acabe as suas tarefas e so depois mata o processo

**this.**processoVaguear**.**waitFor**();**

**}** **catch** **(**InterruptedException e**)** **{**

// TODO Auto-generated catch block

e**.**printStackTrace**();**

**}**

**}**

**}**

O lançamento do *vaguear* conta com uns argumentos adicionais face ao *gestor*, são os que tratam da informação necessária para ligar ao robot, e indicar se o processo *vaguear* está a ser lançado pela *GUI* ou pelo *Gestor*.  
  
Este tipo de lançamento é igualmente aplicado no lançamento do processo evitar.

## **MailBox**

Esta classe deve-se à necessidade da comunicação entre processos ser feita de uma maneira semelhante a uma caixa de correio, em que cada processo pode ler da sua caixa de correio e escrever na dos outros.

public MailBox**(**String nome**)** **{**

**try** **{**

**this.**file **=** **new** File**(**path**+**nome**);**

**this.**file**.**createNewFile**();**

filechannel **=** **new** RandomAccessFile**(this.**file**,** "rw"**).**getChannel**();**

**}** **catch** **(**Exception e**)** **{**

e**.**printStackTrace**();**

**}**

**try** **{**

map **=** filechannel**.**map**(**FileChannel**.**MapMode**.**READ\_WRITE**,** 0**,** MAX\_BUFFER**);**

**}** **catch** **(**Exception e**)** **{**

e**.**printStackTrace**();**

**}**

**}**

A mailbox apenas recebe o nome da caixa de correio que quer aceder, seja para ler ou para escrever, e abre um canal de conexão.

public String read**()** **{**

String str **=** **new** String**();**

char c**;**

map**.**position**(**0**);**

**while** **((**c **=** map**.**getChar**())** **!=** '\0'**)** **{**

str **+=** c**;**

**}**

**return** str**;**

**}**

Método de leitura do ficheiro que nos devolve todo o conteúdo presente no ficheiro em String.

public void eraseContent**()** **{**

write**(**"xxxxxxx"**);**

**}**

public void write**(**String msg**)** **{**

char c**;**

map**.**position**(**0**);**

**for** **(**int i **=** 0**;** i **<** msg**.**length**();** i**++)** **{**

c **=** msg**.**charAt**(**i**);**

map**.**putChar**(**c**);**

**}**

map**.**putChar**(**'\0'**);**

**}**

public void closeChannel**()** **{**

**try** **{**

filechannel**.**close**();**

**}** **catch** **(**Exception e**)** **{**

e**.**printStackTrace**();**

**}**

**}**

Foi criado um método que eliminasse toda a informação presente na caixa de correio, de forma a simular uma caixa de correio real, uma vez que consultamos o conteudo da caixa de correio, retiramos a informação deixando a caixa vazia, metodo esse *eraseContent()*, no nosso sistema de troca de mensagens, a mensagem *xxxxxx* é reconhecida como vazio.

A comunicação entre processos tem a seguinte nomenclatura de prefixos:

- *PV* -> Processo Vaguear  
- *PA* -> Processo Avoid  
- *PG* -> Processo Gestor  
- *PGUI* -> Processo GUI

Para a indicação de qual a acção a fazer no processo, são escritas as seguintes mensagens:

- *sstop* – indica uma ordem de paragem do processo equivalente ao prefixo.  
- *start* – indica uma ordem de começo do processo equivalente ao prefixo.  
- *para* – mensagem especifica da GUI para parar o processo por ela lançado.

Sabendo isto, todas as mensagens presentes na caixa de correio, terão a seguinte formatação:

Mensagem = Prefixo + ordem  
Ex: Pvsstop (Processo Vaguear Parou)

O método de escrita da caixa de correio, começa por escrever sempre da primeira posição do ficheiro.

Apenas se usa o método *closeChannel()* quando quisermos terminar um processo, terminando assim o seu acesso à caixa de correio.

# **Conclusão**

Na realização deste trabalho foram encontrados alguns problemas na implementação da nossa solução, contudo foi possível contornar tais dificuldades de forma a completar os requisitos pedidos.  
Na fase de desenvolvimento do processo *Gestor* que administra os subprocessos *Vaguear* e *Evitar Obstáculo*, o objecto *Runtime* estava ser usado para o lançamento dos subprocessos, tal não estava a ser possível pois o caminho do *ficheiro .jar* não estava a ser reconhecido mesmo estando correto, em solução, optamos por usar o *ProcessBuilder* não tendo assim nenhum erro de execução e os processos lançados corretamente.

Inicialmente em fases de testes, os estados *closeVaguear* e *closeEvitar* estavam dimensionados para serem os estados responsáveis de terminar os subprocessos *Vaguear* e *Evitar Obstáculo* usados, contudo no desenvolvimento da solução, a presença de tais estados era redundante face às verificações que tinham que ser feitas para o fecho de processos acontecer, em solução, uma simples verificação da caixa de correio no fim de cada estado.

Na fase de desenvolvimento da ligação dos processos e a GUI, deparamo-nos com o problema do *evitar obstáculo* não poder fazer uma ligação logo assim que lança o processo, pois o comportamento em si, não faz sentido agir assim que o ligamos mas sim ficar à espera que certa mensagem chegue à sua caixa de correio enviada pela GUI, agregado a isto, implementamos uma verificação do sensor de toque no fim de cada ação enviada para o robot, com ajuda de alguns métodos já fornecidos de cálculo de tempo. A desvantagem desta implementação reside no bloqueamento da GUI sempre que efetuamos esta verificação, com tempo igual ao calculado face à distância percorrida.

A nomenclatura implementada nesta solução tem uns contras encontrados na fase de testes:  
O numero de caracteres de “PVstart” ser diferente de “PVstop”, havendo por vezes restos de mensagens anteriores e podendo interferir na comunicação entre as caixas de correio, visto que os ficheiros nunca são eliminados no fim de cada processo, sendo necessário eliminar remotamente.

Ficaram ainda por completar alguns aspetos referentes á última parte do trabalho, tais como:

- Verificação se há alguma ligação ao robot existente antes do lançamento de qualquer processo;

- Escrita das mensagens dos processos na área de texto da consola de debug da GUI;

# **Bibliografia**

Folhas da disciplina de FSO (Fundamentos de Sistemas Operativos):

https://moodle.isel.pt/pluginfile.php?file=%2F692732%2Fmod\_resource%2Fcontent%2F1%2FFundamentos%20de%20Sistemas%20Operativos.pdf

# 