

# Exercício para Avaliação n.º 3

Bruno Azevedo\*and Miguel Costa†

*Módulo Engenharia Gramatical,  
UCE30 Engenharia de Linguagens,  
Mestrado em Engenharia Informatica,  
Universidade do Minho*

22 de Fevereiro de 2012

## Resumo

Este documento apresenta a resolução do Exercício Prático n.º 3 do módulo de Engenharia Gramatical. O exercício está relacionado com a geração automática de Processadores de Linguagens a partir de Gramáticas.

Era pretendido criar uma linguagem para fazer movimentar um Robo num Terreno e depois criar um processador para as frases da linguagem com algumas funcionalidades.

---

\*Email: azevedo.252@gmail.com

†Email: miguel pintodacosta@gmail.com

# Conteúdo

<b>1</b>	<b>Ambiente de Trabalho</b>	<b>3</b>
<b>2</b>	<b>Descrição do problema</b>	<b>3</b>
<b>3</b>	<b>Criação da linguagem</b>	<b>3</b>
<b>4</b>	<b>Implementação</b>	<b>6</b>
4.1	Decisões Tomadas . . . . .	6
4.2	Classes . . . . .	6
4.3	Linguagem . . . . .	7
<b>5</b>	<b>Conclusões</b>	<b>10</b>
<b>6</b>	<b>Anexos</b>	<b>11</b>
6.1	Classes em Java . . . . .	11
6.1.1	Robo.java . . . . .	11
6.1.2	Terreno.java . . . . .	24
6.1.3	Movimento.java . . . . .	27
6.1.4	Matrix.java . . . . .	29

# 1 Ambiente de Trabalho

Foi necessário usar um Gerador de Compiladores para gerar o nosso próprio compilador, por isso usámos o ANTLR que é também usado nas aulas. Para facilitar o processo de debugging durante a resolução do problema dado, usámos a ferramenta ANTLRWorks, que tem uma interface bastante agradável e simpática para ajudar a resolver problemas desta natureza.

A linguagem de programação adoptada foi o JAVA. De forma a tornar a nossa solução mais legível e estruturada, criámos classes com o auxílio do IDE NetBeans que nos ajudou no desenvolvimento do código JAVA e ainda na criação da sua documentação (javadoc).

# 2 Descrição do problema

Imaginemos um robo com a função de aspirar um terreno de forma retangular. Este terreno tem uma área que é conhecida pelo robo e que acaba por limitar o raio de ação dele.

O robo pode ter definida uma posição inicial e os seus movimentos podem ser em quatro direções diferentes (norte, sul, este e oeste) com um peso associado que representa a distância que se vai deslocar (por exemplo NORTE 4, desloca-se 4 unidades para norte). Tem ainda a opção de estar ligado ou desligado que define se está ativo ou não para aspirar.

Com base na descrição do robo, era pedido:

1. Criar uma linguagem que conseguisse descrever uma rotina possível para o robo. Esta linguagem deve permitir ainda que tenha no início certas definições como a dimensão do terreno e a posição inicial do robo.
2. Depois de definida a linguagem, tínhamos de criar um processador para as possíveis frases que podiam ser geradas com as seguintes funcionalidades:
  - Verificar que o robo não se movimenta para fora da área de limpeza.
  - Calcular a distância (em cm) que o robo percorreu durante a sua rotina.
  - Determinar quantas mudanças de direção foram feitas pelo robo.
  - Determinar a distância média que o robo se desloca por cada movimento.

# 3 Criação da linguagem

Analisando o que era pretendido para descrever a rotina do robo, tentámos criar uma linguagem com uma sintaxe de fácil leitura e sem ambiguidades. Depois de analisar várias alternativas, definimos a linguagem com a seguinte estrutura:

Listing 1: Estrutura da gramática

```
1 ASPIRADOR
2 {
3   DEFINICOES
4   {
5     definicao1; definicao2;
6   }
7   MOVIMENTOS
8   instrucao1;
9   instrucao2;
10  ....
11 }
```

Uma linguagem tem de ter símbolos terminais e neste caso definimos os símbolos:

- DIM
- POS
- LIGAR
- DESLIGAR
- NORTE
- SUL
- ESTE
- OESTE
- ID
- INT

Definindo formalmente a gramática para representar os eventos possíveis do robo, obtemos:

Listing 2: Gramática

```

1  PLTNgrammar robot;
2
3  /*-----
4   *  PARSER RULES
5   *-----*/
6
7  robot
8  @init {
9      terreno = new Terreno();
10     robo = new Robo(terreno);
11 }
12 @after {
13     System.out.println(terreno.toString());
14     System.out.println(robo.toString());
15     System.out.println(robo.toStringEstatisticas());
16
17     Matrix m = new Matrix(robo, terreno);
18     m.setVisible(true);
19 }
20 : 'ASPIRADOR' '{ corpo }'
21 ;
22
23 corpo
24 : 'DEFINICOES' definicoes 'MOVIMENTOS' movimentos
25 ;
26
27 definicoes
28 : '{ dimensao (posicao)? }'
29 | '{ (posicao)? dimensao }'
30 ;
31 dimensao
32 :DIM '=' '(' INT ',' INT ')' ' ';
33 ;
34 posicao
35 :POS '=' '(' INT ',' INT ')' ' ';
36 ;
37
38 movimentos
39 : movimento (movimento)*
40 ;

```

```

41
42 movimento
43     : LIGAR ';'
44     | DESLIGAR ';'
45     | NORTE INT ';'
46     | SUL INT ';'
47     | ESTE INT ';'
48     | OESTE INT ';'
49     ;
50
51 /*-----
52  * LEXER RULES
53  *-----*/
54
55 DIM      : ('d'|'D')('i'|'I')('m'|'M');
56 POS      : ('p'|'P')('o'|'O')('s'|'S');
57
58 LIGAR    : ('l'|'L')('i'|'I')('g'|'G')('a'|'A')('r'|'R');
59 DESLIGAR : ('d'|'D')('e'|'E')('s'|'S')('l'|'L')('i'|'I')('g'|'G')('a'|'A')('r'|'R');
60
61 NORTE    : ('n'|'N')('o'|'O')('r'|'R')('t'|'T')('e'|'E');
62 SUL      : ('s'|'S')('u'|'U')('l'|'L');
63 ESTE     : ('e'|'E')('s'|'S')('t'|'T')('e'|'E');
64 OESTE    : ('o'|'O')('e'|'E')('s'|'S')('t'|'T')('e'|'E');
65
66 ID       : ('a'..'z'|'A'..'Z'|'_' ) ('a'..'z'|'A'..'Z'|'0'..'9'|'_' ) *
67     ;
68
69 INT      : '0'..'9'+
70     ;
71
72 COMMENT
73     : '//' ~('\n'|\r')* '\r'? '\n' {$channel=HIDDEN;}
74     | '/*' ( options {greedy=false;} : . ) * '*/' {$channel=HIDDEN;}
75     ;
76
77 WS       : (
78     | '\t'
79     | '\r'
80     | '\n'
81     ) {$channel=HIDDEN;}
82     ;

```

Depois de gerada a gramática, uma frase que se pode gerar é:

Listing 3: Frase gerada 1

```

1 ASPIRADOR
2 {
3     DEFINICOES
4     {
5         dim = (100 , 150) ; pos = (0 , 0) ;
6     }
7     MOVIMENTOS
8         LIGAR;
9         NORTE 2 ;
10        DESLIGAR ;
11        SUL 10;
12    }

```

Para provar que era uma frase válida, fizemos a sua árvore de derivação:

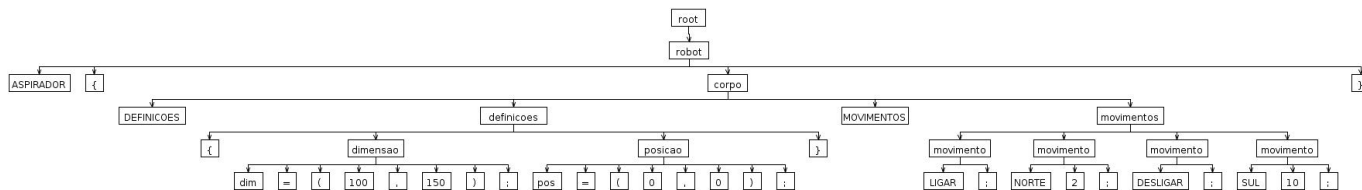


Figura 1: Árvore de derivação

Analisando a árvore gerada, verificámos que o elemento raiz é **robot** e o parser terá de encontrar, no início, a palavra **ASPIRADOR** seguida de um **corpo** que se encontra dentro de chavetas.

O corpo está dividido em 2 partes: **definicoes** e **movimentos**. Nas **definicoes** podemos configurar a **dimensao** do terreno e ainda a **posicao** inicial do robo.

Quanto aos **movimentos**, estes podem ser de 2 tipos, os que fazem realmente movimentar o robo (por exemplo **NORTE 2**) e os que ligam (**LIGAR**) ou desligam (**DESLIGAR**) o robo.

## 4 Implementação

De forma a estruturar melhor todo o exercício, criámos classes em java que nos facilitassem o cálculo de todas as estatísticas e todas as restrições que eram necessárias.

### 4.1 Decisões Tomadas

Como seria de esperar, há pormenores que tinham de ser decididos para colocar o robo no terreno e para o cálculo das estatísticas, algumas decisões tomadas foram:

- Caso não esteja definida a posição inicial do Robo no terreno, é assumido que esta é (0,0), que corresponde ao canto superior esquerdo do terreno.
- Inicialmente, o Robo é colocado no terreno sem direção, assim, apenas depois do primeiro movimento, ele tem a direção definida e é possível contar para efeitos estatísticos a mudança de direção.
- Apenas quando o Robo está no modo ligado é que ele se movimenta, caso contrário ignora todas as instruções que receber, excepto a de **LIGAR**.

### 4.2 Classes

As classes criadas foram:

- Robo
- Terreno
- Movimento
- Matrix

A classe Robo é a responsável por guardar o estado, a posicao atual, a direção atual, todos os movimentos executados pelo robo e por gerar as estatísticas relacionadas com os mesmos. Esta classe contém 4 `ArrayList<Integer>` para guardar inteiros com o valor que foi deslocado em cada uma das direções possíveis e, ainda, um `TreeMap<Integer,Movimento>` em que a `key` corresponde ao número em que o `Movimento` ocorreu, este `value` é do tipo `Movimento` que contém apenas 3 variáveis de instância:

- `Integer num` - número em que o movimento ocorreu.
- `Direcao direcao` - direção em que o movimento foi feito.
- `Integer distancia` - a distancia percorrida nesse movimento.

Este `TreeMap<Integer,Movimento>` é usado apenas para na animação sabermos a ordem em que os movimentos foram feitos e que tipo de deslocação foi feita pelo robo, enquanto que as estatísticas são todas calculadas a partir dos `ArrayList<Integer>` para ser mais eficiente e não termos que estar sempre a percorrer a estrutura em árvore.

`Terreno` é a classe que contém o valor, em cm, de uma unidade de movimento, as dimensões do terreno onde o robo se vai movimentar e verifica se o robo não se quer deslocar para fora dele. Para confirmar visualmente que tudo o que era pedido ao Robo se concretizava, criámos uma interface onde é possível ver a deslocação, passo a passo, do Robo e ainda as estatísticas geradas. Esta interface corresponde à classe `Matrix` que recorre ao Java SWING para criar a animação.

Em anexo está o código java de cada classe.

### 4.3 Linguagem

Depois de criadas as classes em Java, foi necessário adaptar a nossa gramática de forma a realizar o que era pretendido, e instanciámos as três classes `Robo`, `Terreno` e (`Matrix`).

Resultando em:

Listing 4: Linaguem Final

```

1 grammar robot;
2
3 options {
4     language = Java;
5 }
6
7 @header{
8     import Robot.Robo;
9     import Robot.Terreno;
10    import Robot.Matrix;
11 }
12
13 @members{
14     private Robo robo;
15     private Terreno terreno;
16 }
17
18 /*-----
19  *  PARSE RULES
20  *-----*/
21
22 robot
23 @init {
24     terreno = new Terreno(); // instancia o terreno
25     robo = new Robo(terreno); // instancia o robo
26 }
27 @after {
28     System.out.println(terreno.toString()); //Imprime o valor da unidade (em cm), a
        largura e altura do terreno
29     System.out.println(robo.toString()); //Imprime a posicao inicial e a posicao
        final, o estado final, a direcao final, os movimentos executados por direcao,
        o numero de vezes que mudou de direcao, toda a sequencia de movimentos
        executada e o total de movimentos executados

```

```

30      System.out.println(robo.toStringEstatisticas()); // Imprime, por direcao, o
        numero de deslocacoes realizadas, a distancia percorrida (em cm) e a
        distancia media percorrida por movimentacao. Imprime tambem o numero total de
        deslocacoes, a distancia total percorrida, a distancia media percorrida por
        movimentacao e o numero total de mudancas de direcao
31
32      Matrix m = new Matrix(robo, terreno); // instancia a matrix
33      m.setVisible(true);
34  }
35      : 'ASPIRADOR' '{ corpo }'
36      ;
37
38  corpo
39      : 'DEFINICOES' definicoes 'MOVIMENTOS' movimentos
40      ;
41
42  definicoes
43      : '{ dimensao (posicao)? }'
44      | '{ (posicao)? dimensao }'
45      ;
46  dimensao
47      : DIM '=' '(' x=INT{terreno.setLarg(Integer.parseInt(x.getText()));} ',' //
        define a largura do terreno
48          y=INT{terreno.setAlt(Integer.parseInt(y.getText()));} // define a altura
        do terreno
49          ')' ';'
50      ;
51  posicao
52      : POS '=' '(' x=INT { if (terreno.validaPosX(Integer.parseInt(x.getText()))){ robo.
        setPosX(x.getText()); robo.setPosXini(x.getText());} // se a posicao inicial
        do robo no eixo X for valida (ou seja, esta dentro dos limites do terreno)
        entao define a posicao inicial e atual do robo nesse eixo
53          else System.out.println("Posicao inicial invalida.");
54          }
55          ','
56          y=INT { if (terreno.validaPosY(Integer.parseInt(y.getText()))){ robo.
        setPosY(y.getText()); robo.setPosYini(y.getText());} // se a posicao
        inicial do robo no eixo Y for valida entao define a posicao inicial
        e atual do robo nesse eixo
57          else System.out.println("Posicao inicial invalida.");
58          }
59          ')' ';'
60      ;
61
62  movimentos
63      : movimento (movimento)*
64      ;
65
66  movimento
67      : LIGAR ';' {robo.setEstado("LIGADO");} // define o estado do robo como
        Ligado
68      | DESLIGAR ';' {robo.setEstado("DESLIGADO");} // define o estado do robo
        como Desligado
69      | NORTE INT ';' { if (terreno.validaPosY(robo.getPosY() - Integer.parseInt(
        $INT.text))) {robo.movNorte(Integer.parseInt($INT.text));} // se a posicao
        final for valida, entao movimenta o robo para essa posicao
70          else {System.out.println("Movimento NORTE " + $INT.text + " invalido
        por ultrapassar os limites da area de limpeza!");}
71          }
72      | SUL INT ';' {if (terreno.validaPosY(robo.getPosY() + Integer.parseInt(
        $INT.text))) {robo.movSul(Integer.parseInt($INT.text));} // se a posicao

```



```

73         final for valida, entao movimenta o robo para essa posicao
74         else {System.out.println("Movimento SUL "+ $INT.text +" invalido por
75             ultrapassar os limites da area de limpeza!");}
76     }
77     | ESTE INT ';' { if (terreno.validaPosX(robo.getPosX() + Integer.parseInt(
78         $INT.text))) {robo.movEste(Integer.parseInt($INT.text));} // se a posicao
79         final for valida, entao movimenta o robo para essa posicao
80         else {System.out.println("Movimento ESTE "+ $INT.text +" invalido
81             por ultrapassar os limites da area de limpeza!");}
82     }
83     ;
84
85 /*-----
86 * LEXER RULES
87 *-----*/
88
89 DIM      : ('d'|'D')('i'|'I')('m'|'M');
90 POS      : ('p'|'P')('o'|'O')('s'|'S');
91
92 LIGAR    : ('l'|'L')('i'|'I')('g'|'G')('a'|'A')('r'|'R');
93 DESLIGAR : ('d'|'D')('e'|'E')('s'|'S')('l'|'L')('i'|'I')('g'|'G')('a'|'A')('r'|'R');
94
95 NORTE    : ('n'|'N')('o'|'O')('r'|'R')('t'|'T')('e'|'E');
96 SUL      : ('s'|'S')('u'|'U')('l'|'L');
97 ESTE     : ('e'|'E')('s'|'S')('t'|'T')('e'|'E');
98 OESTE    : ('o'|'O')('e'|'E')('s'|'S')('t'|'T')('e'|'E');
99
100 ID       : ('a'..'z'|'A'..'Z'|'_'') ('a'..'z'|'A'..'Z'|'0'..'9'|'_'')*
101
102 INT      : '0'..'9'+
103
104 COMMENT
105 :      '// ' ~('\n'|'\r')* '\r'? '\n' {$channel=HIDDEN;}
106 |      '/*' ( options {greedy=false;} : . )* '*/' {$channel=HIDDEN;}
107 ;
108
109 WS      : ( ' '
110         | '\t'
111         | '\r'
112         | '\n'
113         ) {$channel=HIDDEN;}
114 ;

```

## 5 Conclusões

A resolução deste exercício permitiu perceber melhor a forma como as linguagens podem ser úteis para gerar um programa, que depende do input que irá receber, o resultado final seja o esperado sem ter de estar a alterar o código do programa que é automaticamente gerado.

Um das dificuldades foi perceber como o Antlr fazia o parser das frases de forma a não haver ambiguidade e conseguir na mesma produção termos acesso ao valor de dois símbolos terminais, tal como acontece, por exemplo, quando queremos saber a dimensão do terreno, em que a solução foi inserir labels para o compilador saber qual o valor pretendido.

Serviu de consolidação da matéria dada até agora no módulo de Engenharia de Linguagens, tendo em conta que conseguimos resolver o exercício com sucesso.

## 6 Anexos

### 6.1 Classes em Java

#### 6.1.1 Robo.java

Listing 5: Robo.java

```
1  package Robot;
2
3  import java.util.ArrayList;
4  import java.util.TreeMap;
5
6  /**
7   * Classe Robo que tem todos os metodos para gerir o Robo
8   * @version 1.0
9   * @author Bruno Azevedo, Miguel Costa
10  */
11 public class Robo {
12
13     /**
14      * Funciona como maquina de estados para o Robo,
15      * ou esta ligado ou desligado
16      */
17     public enum Estado {
18
19         LIGADO, DESLIGADO
20     }
21
22     /**
23      * Indica as possiveis direcoes que o robo pode ter,
24      * NULA indica que nao tem uma direcao definida, acontece por exemplo
25      * quando o robo e criado.
26      */
27     public enum Direcao {
28
29         NULA,
30         NORTE,
31         SUL,
32         ESTE,
33         OESTE
34     }
35     private int posx = 0;           // Posicao x por defeito = 0
36     private int posy = 0;           // Posicao y por defeito = 0
37     private int posx_ini = 0;       // Posicao x inicial por
38         defeito = 0
39     private int posy_ini = 0;       // Posicao y inicial por
40         defeito = 0
41     private Estado estado;          // estado do robo (ligado ou
42         desligado)
43     private Direcao dir;            // direcao atual do
44         aspirador
45     private ArrayList<Integer> norte = new ArrayList<Integer>(); // array
46         que armazena os movimentos na direcao norte
47     private ArrayList<Integer> sul = new ArrayList<Integer>();   // array
48         que armazena os movimentos na direcao sul
```

```

43 private ArrayList<Integer> este = new ArrayList<Integer>(); // array
    que armazena os movimentos na direcao este
44 private ArrayList<Integer> oeste = new ArrayList<Integer>(); // array
    que armazena os movimentos na direcao oeste
45 private int mud_dir = 0; // armazena as mudancas de
    direcao
46 private TreeMap<Integer, Movimento> movs = new TreeMap<Integer,
    Movimento>();
47 private int totalMovs = 0;
48 private Terreno terreno;
49
50 /**
51  * Construtor para criar um Robo
52  * @param t Recebe uma referência para o terreno em que vai estar
53  */
54 public Robo(Terreno t) {
55     estado = Estado.DESLIGADO;
56     dir = Direcao.NULA;
57     terreno = t;
58 }
59
60 /**
61  * Devolve o estado do Robot, LIGADO ou DESLIGADO
62  * @return Devolve o resultado do tipo Estado
63  */
64 public Estado getEstado() {
65     return estado;
66 }
67
68 /**
69  * Devolve a direcao atual do robo
70  * @return Devolve o resultado do tipo Direcao.
71  */
72 public Direcao getDirecao() {
73     return dir;
74 }
75
76 /**
77  * Posicao x atual do robo
78  * @return Posicao x atual do Robo (inteiro)
79  */
80 public int getPosX() {
81     return posx;
82 }
83
84 /**
85  * Posicao y atual do robo
86  * @return Posicao y atual do Robo (inteiro)
87  */
88 public int getPosY() {
89     return posy;
90 }
91
92 /**
93  * Posicao x inicial do robo
94  * @return Posicao x inicial do Robo (inteiro)

```

```

95     */
96     public int getPosXini() {
97         return posX_ini;
98     }
99
100    /**
101     * Posicao y inicial do robo
102     * @return Posicao y inicial do Robo (inteiro)
103     */
104    public int getPosYini() {
105        return posy_ini;
106    }
107
108    /**
109     * Devolve um array com todas as deslocacoes para norte
110     * @return ArrayList com as distancias que o robo percorreu para norte
111     */
112    public ArrayList<Integer> getNorte() {
113        ArrayList<Integer> n = new ArrayList<Integer>();
114        for (Integer i : norte) {
115            n.add(i);
116        }
117        return n;
118    }
119
120    /**
121     * Devolve um array com todas as deslocacoes para sul
122     * @return ArrayList com as distancias que o robo percorreu para sul
123     */
124    public ArrayList<Integer> getSul() {
125        ArrayList<Integer> r = new ArrayList<Integer>();
126        for (Integer i : sul) {
127            r.add(i);
128        }
129        return r;
130    }
131
132    /**
133     * Devolve um array com todas as deslocacoes para este
134     * @return ArrayList com as distancias que o robo percorreu para este
135     */
136    public ArrayList<Integer> getEste() {
137        ArrayList<Integer> r = new ArrayList<Integer>();
138        for (Integer i : este) {
139            r.add(i);
140        }
141        return r;
142    }
143
144    /**
145     * Devolve um array com todas as deslocacoes para oeste
146     * @return ArrayList com as distancias que o robo percorreu para oeste
147     */
148    public ArrayList<Integer> getOeste() {
149        ArrayList<Integer> r = new ArrayList<Integer>();
150        for (Integer i : oeste) {

```

```

151         r.add(i);
152     }
153     return r;
154 }
155
156 /**
157  * Devolve os movimentos do robo
158  * @return TreeMap em que a chave e um inteiro que corresponde ao numero
159  * da ordem que o movimento foi feito e o value e do tipo Movimento
160  */
161 public TreeMap<Integer, Movimento> getMovimentos() {
162     TreeMap<Integer, Movimento> r = new TreeMap<Integer, Movimento>();
163
164     for (Movimento m : movs.values()) {
165         r.put(m.getNum(), m.clone());
166     }
167     return r;
168 }
169
170 /**
171  * Dado o numero do movimento que queremos, devolve a classe Movimento
172  * correspondente
173  * @param num Numero do movimento
174  * @return Classe Movimento que corresponde ao numero dado como
175  * parametro
176  */
177 public Movimento getMovimento(int num) {
178     return movs.get(num).clone();
179 }
180
181 /**
182  * Altera a Posicao X do robo.
183  * Usada quando e preciso definir a Posicao inicial e quando a
184  * movimentacao
185  * @param x Nova posicao x em que o robo vai ficar
186  */
187 public void setPosX(int x) {
188     posX = x;
189 }
190
191 /**
192  * Altera a Posicao X do robo recebendo uma string que contem o valor
193  * inteiro
194  * Usada quando e preciso definir a Posicao inicial e quando a
195  * movimentacao
196  * @param x Nova posicao x em que o robo vai ficar
197  */
198 public void setPosX(String x) {
199     posX = Integer.parseInt(x);
200 }
201
202 /**
203  * Altera a Posicao Y do robo
204  * Usada quando e preciso definir a Posicao inicial e quando ha
205  * movimentacao
206  * @param y Nova posicao y em que o robo vai ficar

```

```

201     */
202     public void setPosY(int y) {
203         posy = y;
204     }
205
206     /**
207      * Altera a Posicao Y do robo recebendo uma string que contem o valor
208      * inteiro,
209      * usada quando e preciso definir a Posicao inicial e quando ha
210      * movimentacao
211      * @param y Nova posicao y em que o robo vai ficar
212      */
213     public void setPosY(String y) {
214         posy = Integer.parseInt(y);
215     }
216
217     /**
218      * Altera a Posicao X do robo.
219      * Usada quando e preciso definir a Posicao inicial
220      * @param x Nova posicao x em que o robo tem inicialmente
221      */
222     public void setPosXini(int x) {
223         posx_ini = x;
224     }
225
226     /**
227      * Altera a Posicao X do robo recebendo uma string que contem o valor
228      * inteiro
229      * Usada quando e preciso definir a Posicao inicial
230      * @param x Nova posicao x em que o robo tem inicialmente
231      */
232     public void setPosXini(String x) {
233         posx_ini = Integer.parseInt(x);
234     }
235
236     /**
237      * Altera a Posicao Y do robo
238      * Usada quando e preciso definir a Posicao inicial
239      * @param y Nova posicao y em que o robo tem inicialmente
240      */
241     public void setPosYini(int y) {
242         posy_ini = y;
243     }
244
245     /**
246      * Altera a Posicao Y do robo recebendo uma string que contem o valor
247      * inteiro
248      * Usada quando e preciso definir a Posicao inicial
249      * @param y Nova posicao y em que o robo tem inicialmente
250      */
251     public void setPosYini(String y) {
252         posy_ini = Integer.parseInt(y);
253     }
254
255     /**
256      * Altera o estado recebendo uma variavel do tipo Estado

```

```

253     * @param e
254     */
255     public void setEstado(Estado e) {
256         estado = e;
257     }
258
259     /**
260     * Altera o estado recebendo uma String
261     * @param e
262     */
263     public void setEstado(String e) {
264
265         if (e.equalsIgnoreCase("LIGADO")) {
266             estado = Estado.LIGADO;
267             return;
268         }
269         if (e.equalsIgnoreCase("DESLIGADO")) {
270             estado = Estado.DESLIGADO;
271             return;
272         }
273     }
274
275     /**
276     * Altera a direcao recebendo uma variavel do tipo Direcao
277     * @param d
278     */
279     public void setDirecao(Direcao d) {
280         dir = d;
281     }
282
283     /**
284     * Altera a direcao recebendo uma String
285     * @param d
286     */
287     public void setDirecao(String d) {
288         if (d.equalsIgnoreCase("NORTE")) {
289             dir = Direcao.NORTE;
290             return;
291         }
292         if (d.equalsIgnoreCase("SUL")) {
293             dir = Direcao.SUL;
294             return;
295         }
296         if (d.equalsIgnoreCase("ESTE")) {
297             dir = Direcao.ESTE;
298             return;
299         }
300         if (d.equalsIgnoreCase("OESTE")) {
301             dir = Direcao.OESTE;
302             return;
303         }
304     }
305
306     /**
307     * Move o robo uma certa distancia para Norte
308     * @param dist

```



```

309     */
310     public void movNorte(int dist) {
311
312         // se estiver desligado nao faz nada
313         if (estado == Estado.DESLIGADO) {
314             return;
315         }
316
317         // se a distancia for maior que zero faz coisas, se for menos ou
318         // igual
319         // a zero, ignora e apenas muda a direcao para norte
320         if (dist > 0) {
321             // contabiliza ou nao a mudanca de direccao
322             if (dir != Direcao.NORTE && dir != Direcao.NULA) {
323                 mud_dir++;
324             }
325             // altera o Posicao y
326             posy -= dist;
327             // adiciona a distancia ao array de movimentacoes para norte
328             norte.add(dist);
329             // registra no TreeMap o movimento
330             Movimento m = new Movimento(totalMovs, Movimento.Direcao.NORTE,
331                 dist);
332             movs.put(totalMovs, m);
333             totalMovs++;
334         }
335         // define a direcao para norte
336         dir = Direcao.NORTE;
337     }
338
339     /**
340     * Move o robo uma certa distancia para Sul
341     * @param dist
342     */
343     public void movSul(int dist) {
344
345         // se estiver desligado nao faz nada
346         if (estado == Estado.DESLIGADO) {
347             return;
348         }
349
350         // se a distancia for maior que zero faz coisas, se for menos ou
351         // igual
352         // a zero, ignora e apenas muda a direcao para sul
353         if (dist > 0) {
354             // contabiliza ou nao a mudanca de direccao
355             if (dir != Direcao.SUL && dir != Direcao.NULA) {
356                 mud_dir++;
357             }
358             // altera o Posicao y
359             posy += dist;
360             // adiciona a distancia ao array de movimentacoes para sul
361             sul.add(dist);
362             // registra no TreeMap o movimento
363             Movimento m = new Movimento(totalMovs, Movimento.Direcao.SUL,
364                 dist);
365             movs.put(totalMovs, m);

```

```

361         totalMovs++;
362     }
363     // define a direcao para sul
364     dir = Direcao.SUL;
365 }
366
367 /**
368  * Move o robo para este
369  * @param dist
370  */
371 public void movEste(int dist) {
372     // se estiver desligado nao faz nada
373     if (estado == Estado.DESLIGADO) {
374         return;
375     }
376
377     // se a distancia for maior que zero faz coisas, se for menos ou
378     // igual
379     // a zero, ignora e apenas muda a direcao para este
380     if (dist > 0) {
381         // contabiliza ou nao a mudanca de direccao
382         if (dir != Direcao.ESTE && dir != Direcao.NULA) {
383             mud_dir++;
384         }
385         // altera o Posicao x
386         posX += dist;
387         // adiciona a distancia ao array de movimentacoes para este
388         este.add(dist);
389         // registra no TreeMap o movimento
390         Movimento m = new Movimento(totalMovs, Movimento.Direcao.ESTE,
391             dist);
392         movs.put(totalMovs, m);
393         totalMovs++;
394     }
395     // define a direcao para sul
396     dir = Direcao.ESTE;
397 }
398
399 /**
400  * Move o robo para Oeste
401  * @param dist
402  */
403 public void movOeste(int dist) {
404     // se estiver desligado nao faz nada
405     if (estado == Estado.DESLIGADO) {
406         return;
407     }
408
409     // se a distancia for maior que zero faz coisas, se for menos ou
410     // igual
411     // a zero, ignora e apenas muda a direcao para oeste
412     if (dist > 0) {
413         // contabiliza ou nao a mudanca de direccao
414         if (dir != Direcao.OESTE && dir != Direcao.NULA) {
415             mud_dir++;
416         }
417     }
418 }

```

```

414         // altera o Posicao x
415         posx -= dist;
416         // adiciona a distancia ao array de movimentacoes para este
417         oeste.add(dist);
418         // regista no TreeMap o movimento
419         Movimento m = new Movimento(totalMovs, Movimento.Direcao.OESTE,
            dist);
420         movs.put(totalMovs, m);
421         totalMovs++;
422     }
423     // define a direcao para sul
424     dir = Direcao.OESTE;
425 }
426
427 /**
428  * Mudancas de direcao
429  * @return
430  */
431 public int mudancasDirecao() {
432     return mud_dir;
433 }
434
435 /**
436  * Media do valor que e deslocado para norte
437  * @return
438  */
439 public float mediaDeslocamentoNorte() {
440     if (deslocacoesNorte() == 0) {
441         return 0;
442     }
443     return ((float) totalNorte() / (float) deslocacoesNorte());
444 }
445
446 /**
447  * Media do valor que e deslocado para sul
448  * @return
449  */
450 public float mediaDeslocamentoSul() {
451     if (deslocacoesSul() == 0) {
452         return 0;
453     }
454     return ((float) totalSul() / (float) deslocacoesSul());
455 }
456
457 /**
458  * Media do valor que e deslocado para este
459  * @return
460  */
461 public float mediaDeslocamentoEste() {
462     if (deslocacoesEste() == 0) {
463         return 0;
464     }
465     return ((float) totalEste() / (float) deslocacoesEste());
466 }
467
468 /**

```

```

469     * Media do valor que e deslocado para oeste
470     * @return
471     */
472     public float mediaDeslocamentoOeste() {
473         if (deslocacoesOeste() == 0) {
474             return 0;
475         }
476         return (float) (totalOeste() / deslocacoesOeste());
477     }
478
479     /**
480     * Media da distancia dos deslocamentos feitos pelo robo
481     * @return
482     */
483     public float mediaDeslocamento() {
484         if (totaldeslocacoes() == 0) {
485             return 0;
486         }
487         return ((float) totalDistancias() / (float) totaldeslocacoes());
488     }
489
490     /**
491     * Distancia total da movimentacao do robo para norte
492     * @return
493     */
494     public int totalNorte() {
495         int total = 0;
496         for (Integer i : norte) {
497             total += i;
498         }
499         return total * terreno.getUni();
500     }
501
502     /**
503     * Distancia total da movimentacao do robo para sul
504     * @return
505     */
506     public int totalSul() {
507         int total = 0;
508         for (Integer i : sul) {
509             total += i;
510         }
511         return total * terreno.getUni();
512     }
513
514     /**
515     * Distancia total da movimentacao do robo para este
516     * @return
517     */
518     public int totalEste() {
519         int total = 0;
520         for (Integer i : este) {
521             total += i;
522         }
523         return total * terreno.getUni();
524     }

```

```

525
526 /**
527  * Distancia total da movimentacao do robo para oeste
528  * @return
529  */
530 public int totalOeste() {
531     int total = 0;
532     for (Integer i : oeste) {
533         total += i;
534     }
535     return total * terreno.getUni();
536 }
537
538 /**
539  * Distancia total da movimentacao do robo para todas as direcoes
540  */
541 public int totalDistancias() {
542     return totalNorte() + totalSul() + totalEste() + totalOeste();
543 }
544
545 /**
546  * Numero de vezes que o robo se deslocou para norte
547  * @return
548  */
549 public int deslocacoesNorte() {
550     return norte.size();
551 }
552
553 /**
554  * Numero de vezes que o robo se deslocou para sul
555  * @return
556  */
557 public int deslocacoesSul() {
558     return sul.size();
559 }
560
561 /**
562  * Numero de vezes que o robo se deslocou para este
563  * @return
564  */
565 public int deslocacoesEste() {
566     return este.size();
567 }
568
569 /**
570  * Numero de vezes que o robo se deslocou para oeste
571  * @return
572  */
573 public int deslocacoesOeste() {
574     return oeste.size();
575 }
576
577 /**
578  * Numero de vezes que o robo se deslocou para todas as direcoes
579  * @return
580  */

```

```

581 public int totaldeslocacoes() {
582     return deslocacoesNorte() + deslocacoesSul() + deslocacoesEste() +
        deslocacoesOeste();
583 }
584
585 /**
586  * metodo toString, mostra toda a informacao do Robo
587  * @return
588  */
589 @Override
590 public String toString() {
591     return "Robo{" + "posx=" + posx + ", posy=" + posy + ", posx_ini=" +
        posx_ini + ", posy_ini=" + posy_ini + ", estado=" + estado + ",
        dir=" + dir + ", norte=" + norte + ", sul=" + sul + ", este=" +
        este + ", oeste=" + oeste + ", mud_dir=" + mud_dir + ", movs=" +
        movs + ", totalMovs=" + totalMovs + '}';
592 }
593
594 /**
595  * Devolve uma String com todas as estatisticas do Robo
596  * @return
597  */
598 public String toStringEstatisticas() {
599     StringBuilder s = new StringBuilder("ESTATISTICAS\n");
600     s.append("    Norte:\n");
601     s.append("        Total deslocacoes: ").append(deslocacoesNorte()).
        append("\n");
602     s.append("        Total distancia percorrida: ").append(totalNorte())
        .append("\n");
603     s.append("        Media de distancia percorrida por cada
        movimentacao: ").append(mediaDeslocamentoNorte()).append("\n");
604
605     s.append("    Sul:\n");
606     s.append("        Total deslocacoes: ").append(deslocacoesSul()).
        append("\n");
607     s.append("        Total distancia percorrida: ").append(totalSul()).
        append("\n");
608     s.append("        Media de distancia percorrida por cada
        movimentacao: ").append(mediaDeslocamentoSul()).append("\n");
609
610     s.append("    Este:\n");
611     s.append("        Total deslocacoes: ").append(deslocacoesEste()).
        append("\n");
612     s.append("        Total distancia percorrida: ").append(totalEste())
        .append("\n");
613     s.append("        Media de distancia percorrida por cada
        movimentacao: ").append(mediaDeslocamentoEste()).append("\n");
614
615     s.append("    Oeste:\n");
616     s.append("        Total deslocacoes: ").append(deslocacoesOeste()).
        append("\n");
617     s.append("        Total distancia percorrida: ").append(totalOeste())
        .append("\n");
618     s.append("        Media de distancia percorrida por cada
        movimentacao: ").append(mediaDeslocamentoOeste()).append("\n");
619

```

```

620     s.append("        TOTAL:\n");
621     s.append("        Total deslocacoes: ").append(totaldeslocacoes()).
        append("\n");
622     s.append("        Total distancia percorrida: ").append(
        totalDistancias()).append("\n");
623     s.append("        Media de distancia percorrida por cada
        movimentacao: ").append(mediaDeslocamento()).append("\n");
624     s.append("        Total mudancas direcao: ").append(mudancasDirecao
        ()).append("\n");
625
626     return s.toString();
627 }
628 }

```

### 6.1.2 Terreno.java

Listing 6: Terreno.java

```
1  package Robot;
2
3  /**
4   * Classe Terreno que contem as dismensoes do terreno e verifica se o Robo
      esta a deslocar-se dentro deste.
5   * @author miguel
6   */
7  public class Terreno {
8
9      private int uni;
10     private int larg;
11     private int alt;
12
13     /**
14      * Construtor sem dimensoes, por defeito coloca: uni=25; larg=100; alt
        =100
15     */
16     public Terreno() {
17         this.uni = 25;
18         this.larg = 100;
19         this.alt = 100;
20     }
21
22     /**
23      * Constutor que recebe como parametro a dimensao do terreno. Coloca a
        uni a 25
24     * @param larg
25     * @param alt
26     */
27     public Terreno(int larg, int alt) {
28         this.uni = 25;
29         this.larg = larg;
30         this.alt = alt;
31     }
32
33     /**
34      * Devolve a altura do terreno
35     * @return
36     */
37     public int getAlt() {
38         return alt;
39     }
40
41     /**
42      * Altera a altura do terreno
43     * @param alt
44     */
45     public void setAlt(int alt) {
46         this.alt = alt;
47     }
48
49     /**
50      * Devolve a largura do terreno
```



```

51     * @return
52     */
53     public int getLarg() {
54         return larg;
55     }
56
57     /**
58     * Altera a largura do terreno
59     * @param larg
60     */
61     public void setLarg(int larg) {
62         this.larg = larg;
63     }
64
65     /**
66     * Devolve a quando corresponde uma unidade em cm
67     * @return
68     */
69     public int getUni() {
70         return uni;
71     }
72
73     /**
74     * Altera o valor a que uma unidade corresponde em cm
75     * @param uni
76     */
77     public void setUni(int uni) {
78         this.uni = uni;
79     }
80
81     /**
82     * Verifica se uma posicao esta ou nao dentro do terreno no valor x
83     * @param posx
84     * @return
85     */
86     public boolean validaPosX(int posx) {
87         if (posx >= larg || posx < 0) {
88             return false;
89         }
90         return true;
91     }
92
93     /**
94     * Verifica se uma posicao esta dentro ou nao do terreno no valor y
95     * @param posy
96     * @return
97     */
98     public boolean validaPosY(int posy) {
99         if (posy >= alt || posy < 0) {
100             return false;
101         }
102         return true;
103     }
104
105     /**
106     * Devolve a informacao do terreno

```

```
107     * @return
108     */
109     @Override
110     public String toString() {
111         return "Terreno{" + "uni=" + uni + ", larg=" + larg + ", alt=" + alt
112             + '}';
113     }
114 }
```

### 6.1.3 Movimento.java

Listing 7: Movimento.java

```
1  package Robot;
2
3  /**
4   * Classe que guarda o registo de um movimento
5   * @author Bruno Azevedo, Miguel Costa
6   */
7  public class Movimento {
8
9      public enum Direcao {
10
11          NULA,
12          NORTE,
13          SUL,
14          ESTE,
15          OESTE
16      }
17      private Integer num;
18      private Direcao direcao;
19      private Integer distancia;
20
21      public Movimento(Integer n, Direcao d, Integer dist) {
22          num = n;
23          direcao = d;
24          distancia = dist;
25      }
26
27      private Movimento(Movimento m) {
28          num = m.getNum();
29          direcao = m.getDirecao();
30          distancia = m.getDistancia();
31      }
32
33      public Direcao getDirecao() {
34          return direcao;
35      }
36
37      public Integer getDistancia() {
38          return distancia;
39      }
40
41      public Integer getNum() {
42          return num;
43      }
44
45      @Override
46      public Movimento clone() {
47          return new Movimento(this);
48      }
49
50      @Override
51      public String toString() {
52          return "Movimento{" + "num=" + num + ", direcao=" + direcao + ",
              distancia=" + distancia + '}';
```

53 }  
54 }

#### 6.1.4 Matrix.java

Listing 8: Matrix.java

```
1  /*
2   * Matrix.java
3   *
4   * Created on 2/Fev/2012, 17:33:48
5   */
6  package Robot;
7
8  import java.awt.Color;
9  import java.util.logging.Level;
10 import java.util.logging.Logger;
11 import javax.swing.JOptionPane;
12 import javax.swing.JPanel;
13
14 /**
15  * Classe que para a interface
16  * @author Bruno Azevedo, Miguel Costa
17  */
18 public final class Matrix extends javax.swing.JFrame {
19
20     JPanel[] paneis;
21     Robo _r;
22     Terreno _t;
23
24     /** Creates new form Matrix */
25     public Matrix(Robo r, Terreno t) {
26         initComponents();
27
28         _r = r;
29         _t = t;
30         defineTerreno();
31     }
32
33     /**
34      * Cria uma grelha a que corresponde o terreno
35      */
36     public void defineTerreno() {
37         int l = _t.getLarg();
38         int a = _t.getAlt();
39         int dim = l * a;
40         paneis = new JPanel[dim];
41         jPanelMatrix.setBackground(new java.awt.Color(254, 254, 254));
42         jPanelMatrix.setLayout(new java.awt.GridLayout(_t.getLarg(), _t.
            getAlt()));
43
44         for (int i = 0; i < paneis.length; i++) {
45             JPanel j = new JPanel();
46             j.setBackground(new java.awt.Color(254, 254, 254));
47             j.setBorder(javax.swing.BorderFactory.createLineBorder(new java.
                awt.Color(0, 0, 0)));
48             javax.swing.GroupLayout layout = new javax.swing.GroupLayout(j);
49             j.setLayout(layout);
50             layout.setHorizontalGroup(layout.createParallelGroup(javax.swing.
                GroupLayout.Alignment.LEADING).addGroup(0, 269, Short.
```

```

MAX_VALUE));
51
52 layout.setVerticalGroup(layout.createParallelGroup(javax.swing.
    GroupLayout.Alignment.LEADING).addGap(0, 86, Short.MAX_VALUE)
    );
53
54 jPanelMatrix.add(j);
55 paneis[i] = j;
56 }
57
58 }
59
60 /**
61  * Pinta o caminho do robo no terreno
62  */
63 public void caminho() {
64     int posx_atual = _r.getPosXini();
65     int posy_atual = _r.getPosYini();
66
67     pinta_ini();
68
69     for (Integer i : _r.getMovimentos().keySet()) {
70         Movimento v = _r.getMovimento(i);
71         //System.out.println(v.toString());
72         if (v.getDirecao() == Movimento.Direcao.NORTE) {
73             int d = v.getDistancia();
74             //pinta(posx_atual, posy_atual);
75             for (int j = 0; j < d; j++) {
76                 posy_atual--;
77                 pinta(posx_atual, posy_atual);
78             }
79         } else if (v.getDirecao() == Movimento.Direcao.SUL) {
80             int d = v.getDistancia();
81             //pinta(posx_atual, posy_atual);
82             for (int j = 0; j < d; j++) {
83                 posy_atual++;
84                 pinta(posx_atual, posy_atual);
85             }
86         } else if (v.getDirecao() == Movimento.Direcao.ESTE) {
87             int d = v.getDistancia();
88             //pinta(posx_atual, posy_atual);
89             for (int j = 0; j < d; j++) {
90                 posx_atual++;
91                 pinta(posx_atual, posy_atual);
92             }
93         } else if (v.getDirecao() == Movimento.Direcao.OESTE) {
94             int d = v.getDistancia();
95             //pinta(posx_atual, posy_atual);
96             for (int j = 0; j < d; j++) {
97                 posx_atual--;
98                 pinta(posx_atual, posy_atual);
99             }
100         }
101     }
102
103 }

```

```

104         pinta_ini();
105         pinta_fim(posx_atual, posy_atual);
106
107     }
108
109     /**
110     * Devolve a posicao no array a que corresponde o x,y
111     * @param x
112     * @param y
113     * @return
114     */
115     private int posicao(int x, int y) {
116         try {
117             return _t.getLarg() * y + x;
118         } catch (Exception e) {
119             return 0;
120         }
121     }
122
123     /**
124     * Pinta uma celula na grelha do terreno
125     * @param x
126     * @param y
127     */
128     private void pinta(int x, int y) {
129         paneis[posicao(x, y)].setBackground(new java.awt.Color(10, 10, 254))
130         ;
131         this.update(this.getGraphics());
132         //this.update(paneis[posicao(x, y)].getGraphics());
133         try {
134             Thread.sleep(500);
135         } catch (InterruptedException ex) {
136             Logger.getLogger(Matrix.class.getName()).log(Level.SEVERE, null,
137                 ex);
138         }
139     }
140
141     /**
142     * Pinta a celula correspondente a posicao inicial do robo
143     */
144     private void pinta_ini() {
145         int x = _r.getPosXini();
146         int y = _r.getPosYini();
147         paneis[posicao(x, y)].setBackground(new java.awt.Color(10, 254, 10))
148         ;
149         this.update(this.getGraphics());
150         try {
151             Thread.sleep(500);
152         } catch (InterruptedException ex) {
153             Logger.getLogger(Matrix.class.getName()).log(Level.SEVERE, null,
154                 ex);
155         }
156     }
157
158     /**
159     * Pinta a celula correspondente a posicao final do robo

```

```

156     * @param x
157     * @param y
158     */
159     private void pinta_fim(int x, int y) {
160         paneis[posicao(x, y)].setBackground(new java.awt.Color(254, 10, 10))
161         ;
162         this.update(this.getGraphics());
163         try {
164             Thread.sleep(500);
165         } catch (InterruptedException ex) {
166             Logger.getLogger(Matrix.class.getName()).log(Level.SEVERE, null,
167                 ex);
168         }
169     }
170
171     /** This method is called from within the constructor to
172     * initialize the form.
173     * WARNING: Do NOT modify this code. The content of this method is
174     * always regenerated by the Form Editor.
175     */
176     @SuppressWarnings("unchecked")
177     // <editor-fold defaultstate="collapsed" desc="Generated Code">//GEN-
178     BEGIN initComponents
179     private void initComponents() {
180
181         jPanelMatrix = new javax.swing.JPanel();
182         jButtonStart = new javax.swing.JButton();
183         jButtonStats = new javax.swing.JButton();
184
185         setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
186
187         jPanelMatrix.setBackground(new java.awt.Color(254, 254, 254));
188         jPanelMatrix.setLayout(new java.awt.GridLayout(3, 3));
189
190         jButtonStart.setText("Iniciar");
191         jButtonStart.addActionListener(new java.awt.event.ActionListener() {
192             public void actionPerformed(java.awt.event.ActionEvent evt) {
193                 jButtonStartActionPerformed(evt);
194             }
195         });
196
197         jButtonStats.setText("Estatísticas");
198         jButtonStats.addActionListener(new java.awt.event.ActionListener() {
199             public void actionPerformed(java.awt.event.ActionEvent evt) {
200                 jButtonStatsActionPerformed(evt);
201             }
202         });
203
204         javax.swing.GroupLayout layout = new javax.swing.GroupLayout(
205             getContentPane());
206         getContentPane().setLayout(layout);
207         layout.setHorizontalGroup(
208             layout.createParallelGroup(javax.swing.GroupLayout.Alignment.
209                 LEADING)
210             .addGroup(layout.createSequentialGroup()
211                 .addContainerGap(

```



```

207         .addGroup(layout.createParallelGroup(javax.swing.GroupLayout
208             .Alignment.LEADING)
209             .addGroup(layout.createSequentialGroup()
210                 .addComponent(jButtonStart, javax.swing.GroupLayout.
211                     PREFERRED_SIZE, 113, javax.swing.GroupLayout.
212                         PREFERRED_SIZE)
213                 .addPreferredGap(javax.swing.LayoutStyle.
214                     ComponentPlacement.RELATED, 298, Short.MAX_VALUE)
215                 .addComponent(jButtonStats, javax.swing.GroupLayout.
216                     PREFERRED_SIZE, 158, javax.swing.GroupLayout.
217                         PREFERRED_SIZE))
218                 .addComponent(jPanelMatrix, javax.swing.GroupLayout.
219                     DEFAULT_SIZE, 569, Short.MAX_VALUE))
220             .addContainerGap())
221     );
222     layout.setVerticalGroup(
223         layout.createParallelGroup(javax.swing.GroupLayout.Alignment.
224             LEADING)
225         .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, layout.
226             createSequentialGroup()
227                 .addContainerGap()
228                 .addComponent(jPanelMatrix, javax.swing.GroupLayout.
229                     DEFAULT_SIZE, 414, Short.MAX_VALUE)
230                 .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.
231                     RELATED)
232                 .addGroup(layout.createParallelGroup(javax.swing.GroupLayout
233                     .Alignment.BASELINE)
234                     .addComponent(jButtonStart)
235                     .addComponent(jButtonStats))
236                 .addContainerGap())
237     );
238     pack();
239 }// </editor-fold>//GEN-END:initComponents
240
241 /**
242  * Evento de clicar no botao start
243  * @param evt
244  */
245 private void jButtonStartActionPerformed(java.awt.event.ActionEvent evt)
246 {
247     //GEN-FIRST:event_jButtonStartActionPerformed
248     caminho();
249 }//GEN-LAST:event_jButtonStartActionPerformed
250
251 /**
252  * Evento de clicar no botao para ver as estatisticas
253  * @param evt
254  */
255 private void jButtonStatsActionPerformed(java.awt.event.ActionEvent evt)
256 {
257     //GEN-FIRST:event_jButtonStatsActionPerformed
258     JOptionPane.showMessageDialog(rootPane, _r.toStringEstatisticas());
259 }//GEN-LAST:event_jButtonStatsActionPerformed
260
261 /**
262  * @param args the command line arguments
263  */

```

```

249 public static void main(String args[]) {
250     /* Set the Nimbus look and feel */
251     //<editor-fold defaultstate="collapsed" desc=" Look and feel setting
        code (optional) ">
252     /* If Nimbus (introduced in Java SE 6) is not available, stay with
        the default look and feel.
253     * For details see http://download.oracle.com/javase/tutorial/
        uiswing/lookandfeel/plaf.html
254     */
255     try {
256         for (javax.swing.UIManager.LookAndFeelInfo info : javax.swing.
            UIManager.getInstalledLookAndFeels()) {
257             if ("Nimbus".equals(info.getName())) {
258                 javax.swing.UIManager.setLookAndFeel(info.getClassName()
                );
259                 break;
260             }
261         }
262     } catch (ClassNotFoundException ex) {
263         java.util.logging.Logger.getLogger(Matrix.class.getName()).log(
            java.util.logging.Level.SEVERE, null, ex);
264     } catch (InstantiationException ex) {
265         java.util.logging.Logger.getLogger(Matrix.class.getName()).log(
            java.util.logging.Level.SEVERE, null, ex);
266     } catch (IllegalAccessException ex) {
267         java.util.logging.Logger.getLogger(Matrix.class.getName()).log(
            java.util.logging.Level.SEVERE, null, ex);
268     } catch (javax.swing.UnsupportedLookAndFeelException ex) {
269         java.util.logging.Logger.getLogger(Matrix.class.getName()).log(
            java.util.logging.Level.SEVERE, null, ex);
270     }
271     //</editor-fold>
272
273     /* Create and display the form */
274     java.awt.EventQueue.invokeLater(new Runnable() {
275
276         @Override
277         public void run() {
278             new Matrix(null, null).setVisible(true);
279         }
280     });
281 }
282 // Variables declaration - do not modify//GEN-BEGIN:variables
283 private javax.swing.JButton jButtonStart;
284 private javax.swing.JButton jButtonStats;
285 private javax.swing.JPanel jPanelMatrix;
286 // End of variables declaration//GEN-END:variables
287 }

```