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

13 de Março de 2012

Resumo

Este documento apresenta as resoluções dos Exercícios Práticos n.º 3 e n.º 4 do módulo de Engenharia Gramatical. O exercício está relacionado com a geração automática de Processadores de Linguagens a partir de Gramáticas.

Para o exercício n.º 3 era pretendido utilizar a gramática `Genea` j´a utilizada nas aulas para calcular algumas estatísticas relacionadas com ela. Para o exercício 4 o objectivo era 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: `miguelpintodacosta@gmail.com`

Conteúdo

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 Exercício n.º 3 - Genea

2.1 Descrição do problema

O pretendido para este exercício era calcular algumas estatísticas a partir de uma frase válida para a linguagem do `Genea`, os cálculos efectuados foram:

- Total de Famílias
- Total de Progenitores
- Total de Filhos
- Total de filhos de cada família
- Média de filhos por família

Além das estatísticas é verificado ainda se as datas são válidas, verificámos se:

- a data de casamento é posterior à data de nascimento do casal;
- a data de morte, nascimento e casamento são datas e não uma string que possa representar outra coisa.

Relembrando uma frase válida para o Genea:

```
PROGENITORES ( 28-02-1988 )
    PAI Antonio, Costa 09-03-1961
    MAE Maria, Costa 21-07-1962
    FILHOS
        Miguel 28-03-1990,
        Pedro 06-04-1992,
        Cristina 02-01-1997
```

que tem como árvore de derivação:

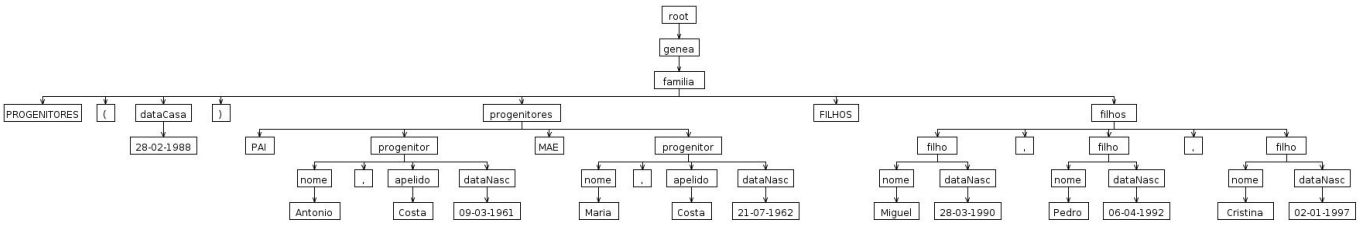

Figura 1: Árvore de derivação de uma frase da Gramática Genea

2.2 Resolução do Problema

Para calcular e verificar o que era pretendido, criámos variáveis globais (na secção members). Como variáveis temos então:

- `int total_progenitores;` // variável que conta o total de progenitores
- `int total_filhos;` // variável que conta o total de filhos.
- `Integer fil_temp;` // variável temporário que contêm o total de filhos de uma família, quando começa uma nova família é coloca novamente a 0.
- `Integer total_familias;` // conta o total de famílias existentes na frase.
- `Integer media_filhos;` // indica o número médio de filhos que as famílias tem.
- `ArrayList<Integer> filhos = new ArrayList<Integer>();` // Array que contêm o número de filhos que cada família tem.

Para o tratamento das datas foi necessário ter as variáveis:

- `GregorianCalendar dataCasa_tmp;` // variável que tem a data de casamento dos progenitores.
- `GregorianCalendar dataNasc1_tmp;` // data de nascimento de um dos progenitores.
- `GregorianCalendar dataNasc2_tmp;` // data de nascimento de um dos progenitores
- `boolean vez = false;` // para controloar se a data a ser lida é a de casamento ou de nascimento.

Para além das variáveis, tivemos ainda de criar algumas funções para verificar as datas:

Listing 1: Funções para as datas

```
/**
* Verifica se uma data é válida.
*/
public String verificaData(String data){

        try{
            String[] valores;
            String delimiter = "-";
            valores = data.split(delimiter);

            Integer a = Integer.parseInt(valores[2]);
            if(a< 1000 || a > 2100){
                return "Ano Invalido";
            }

            Integer m = Integer.parseInt(valores[1]);
            if(m < 1 || m > 12){
                return "Mes Invalido";
            }

            Integer d = Integer.parseInt(valores[0]);
            if(d<1 || d > 31){
                return "Dia invalido";
            }
        } catch(Exception e){
            System.out.println("Erro ao validar data.");
        }
```

```
28          return "";
29      }
30
31      /**
32      * Dada uma string extrai o ano
33      */
34      public Integer getAno(String data){
35          String[] valores;
36          String delimiter = "-";
37          valores = data.split(delimiter);
38          return Integer.parseInt(valores[2]);
39      }
40
41      /**
42      * Dada uma string extrai o mes
43      */
44      public Integer getMes(String data){
45          String[] valores;
46          String delimiter = "-";
47          valores = data.split(delimiter);
48          return Integer.parseInt(valores[1]);
49      }
50
51      /**
52      * Dada uma string extrai o dia
53      */
54      public Integer getDia(String data){
55          String[] valores;
56          String delimiter = "-";
57          valores = data.split(delimiter);
58          return Integer.parseInt(valores[0]);
59      }
```

Depois de criadas todas as funções e variáveis necessárias tivemos de adicionar nas produções as regras de cálculo.

- `int total_progenitores;`
  Para calcular o total de progenitores tivemos de adicionar na produção:
  `progenitor : nome ',' apelido dataNasc dataMorte? ;`
  a instrução:
  `total_progenitores++;`

- `int total_filhos;`
  No cálculo do total de filhos, o comportamento é semelhante ao total de progenitores, mas neste caso, adicionámos na secção init da produção:
  `filho : nome dataNasc dataMorte? ;`
  a instrução:
  `total_filhos++;`

- `Integer total_familias;`
  O total de famílias funciona da mesma forma que as anteriores, em que na secção da produção:
  `familia : 'PROGENITORES' '(' dataCasa ')' progenitores 'FILHOS' filhos? ;`
  foi adicionada a instrução:
  `total_familias++;`

- `ArrayList<Integer> filhos = new ArrayList<Integer>();`
  O array que contém os filhos de cada família, utiliza a variável temporária `Integer fil_temp;`, que conta

os filhos de uma família e é colocada novamente a 0 quando é encontrada uma nova família, para no final (secção after) da produção `filhos : filho (',' filho)*;` ser adicionado mais um índice ao array com o valor correspondente ao número de filhos da família.

- `Integer media_filhos;`
  A média de filhos é calculada no final de toda a frase ter sido reconhecida, ou seja, na secção after da produção `genea : familia+;`. A instrução usada apenas pega no total de filhos e divide pelo total de famílias que já estão calculados (`media_filhos = total_filhos/total_familias;`).

Relativamente às datas, a verificação é feita na produção de cada uma das datas, por exemplo, na produção `dataCasa : DATA ;` foram adicionadas as instruções:
`String data = verificaData(DATA.text); System.out.println(data);` em que a string data guarda algum erro caso não seja válida.

Para verificar se a data de casamento é posterior à data de nascimento dos progenitores, na produção da `familia` são colocadas as instruções para guardar na variável temporária `dataCasa_tmp` a data de casamento do casal:
`String d = dataCasa.text;dataCasa_tmp = new GregorianCalendar(getAno(d), getMes(d), getDia(d));`
As datas de nascimento dos progenitores são criadas da mesma forma que a data do casamento, mas de forma a ir para variáveis diferentes o `PAI` e a `MAE`, é usada a flag `vez`.

Depois de termos as datas todas, no final da produção `familia` é verificado se as datas estão correctas:

```
if(dataNasc1_tmp.after(dataCasa_tmp) || dataNasc2_tmp.after(dataCasa_tmp)){
    System.out.println("Data de Casamento inválida.");
}else {
    System.out.println("Data de casamento válida.");
}
```

2.3 Resultado Final

Depois de correr a gramática em que o input é a frase dada como exemplo anterioremente, obtemos como output:

```
Data de casamento válida.
Total de familias: 1
Total de progenitores: 2
Média de filhos por familia: 3
Filhos da familia 1: 3
```

2.4 Gramática Final

No final de termos a nossa Gramática defenida e com todas as instruções necessários para atingir os objectivos, ficamos com:

Listing 2: Linaguagem Final

```
1  grammar genea;
2
3  @header{
4      import java.util.ArrayList;
5      import java.util.GregorianCalendar;
6  }
7
8  @members{
9      private int total_progenitores = 0;
10     private int total_filhos = 0;
```

```
11      private Integer fil_temp = 0;
12      private Integer total_familias = 0;
13      private Integer media_filhos = 0;
14      private ArrayList<Integer> filhos = new ArrayList<Integer>();
15      // para verificar se as dastas estao correctas
16      private GregorianCalendar dataCasa_tmp;
17      private GregorianCalendar dataNasc1_tmp;
18      private GregorianCalendar dataNasc2_tmp;
19      private boolean vez = false; // para ver se vai para a segunda data ou ainda para
            a primeira
20
21      public String verificaData(String data){
22          try{
23              String[] valores;
24              String delimiter = "-";
25              valores = data.split(delimiter);
26              Integer a = Integer.parseInt(valores[2]);
27              if(a< 1000 || a > 2100){
28                  return "Ano Invalido";
29              }
30              Integer m = Integer.parseInt(valores[1]);
31              if(m < 1 || m > 12){
32                  return "Mes Invalido";
33              }
34              Integer d = Integer.parseInt(valores[0]);
35              if(d<1 || d > 31){
36                  return "Dia invalido";
37              }
38          } catch(Exception e){
39              System.out.println("Erro ao validar data.");
40          }
41          return "";
42      }
43
44      public Integer getAno(String data){
45          String[] valores;
46          String delimiter = "-";
47          valores = data.split(delimiter);
48          return Integer.parseInt(valores[2]);
49      }
50
51      public Integer getMes(String data){
52          String[] valores;
53          String delimiter = "-";
54          valores = data.split(delimiter);
55          return Integer.parseInt(valores[1]);
56      }
57
58      public Integer getDia(String data){
59          String[] valores;
60          String delimiter = "-";
61          valores = data.split(delimiter);
62          return Integer.parseInt(valores[0]);
63      }
64
65      }
66
67  genea
68  @init   {
69      total_progenitores = 0;
70      total_familias = 0;
```

```
71      total_filhos = 0;
72  }
73  @after  {
74      System.out.println("Total de familias: "+total_familias);
75      System.out.println("Total de progenitores: "+total_progenitores);
76      media_filhos = total_filhos/total_familias;
77      System.out.println("Media de filhos por familia: "+media_filhos);
78      int i = 1;
79      for(Integer n : filhos){
80          System.out.println("Filhos da familia "+i +": "+n);
81          i++;
82      }
83  }
84      : familia+
85      ;
86
87  familia
88  @init   {
89      total_familias++;
90  }
91  @after  {
92      if(dataNasc1_tmp.after(dataCasa_tmp) || dataNasc2_tmp.after(dataCasa_tmp)){
93          System.out.println("Data de Casamento invalida.");
94      }else {
95          System.out.println("Data de casamento valida.");
96      }
97  }
98      : 'PROGENITORES' '(' dataCasa ')'   {
99                          String d = $dataCasa.text;
100                         dataCasa_tmp = new GregorianCalendar(getAno(d), getMes(d),
                                getDia(d));
101                         vez = false;
102                         }
103      progenitores 'FILHOS' filhos?
104     ;
105
106 progenitores
107     : 'PAI' progenitor 'MAE' progenitor
108     | 'MAE' progenitor 'PAI' progenitor
109     ;
110
111 progenitor
112     : nome ',' apelido dataNasc     {
113                     total_progenitores++;
114                     if(!vez){
115                         String d = $dataNasc.text;
116                         dataNasc1_tmp = new GregorianCalendar(getAno(d), getMes(d),
                                getDia(d));
117                         vez = true;
118                     }else{
119                         String d = $dataNasc.text;
120                         dataNasc2_tmp = new GregorianCalendar(getAno(d), getMes(d),
                                getDia(d));
121                     }
122                     }
123
124     dataMorte?
125     ;
126
127 filhos
128 @init   {
```

```
129     fil_temp = 0;
130 }
131 @after  {
132     filhos.add(fil_temp);
133 }
134     : filho (',' filho)*
135     ;
136
137 filho
138 @init   {
139     fil_temp++;
140     total_filhos++;
141 }
142     : nome dataNasc dataMorte?
143     ;
144
145 nome    : ID
146     ;
147
148 apelido
149     : ID
150     ;
151
152 dataCasa
153     : DATA  {
154         String data = verificaData($DATA.text);
155         System.out.println(data);
156         }
157     ;
158
159 dataNasc
160     : DATA {
161         String valido = verificaData($DATA.text);
162         System.out.println(valido);
163         }
164     ;
165
166 dataMorte
167     : DATA {
168         String valido = verificaData($DATA.text);
169         System.out.println(valido);
170         }
171     ;
172
173
174 DATA    : INT '-' INT '-' INT
175     ;
176
177 ID  :   ('a'..'z'|'A'..'Z'|'_') ('a'..'z'|'A'..'Z'|'0'..'9'|'_')*
178     ;
179
180 INT :   '0'..'9'+
181     ;
182
183 COMMENT
184     :   '//' ~('\n'|'\r')* '\r'? '\n' {$channel=HIDDEN;}
185     |   '/*' ( options {greedy=false;} : . )* '*/' {$channel=HIDDEN;}
186     ;
187
188 WS  :   ( ' '
189         | '\t'
```

```
190         | '\r'
191         | '\n'
192         ) {$channel=HIDDEN;}
193     ;
```

3 Exercício n.º 4 - Robo

3.1 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.

3.2 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 3: Estrutura da gramática

```
ASPIRADOR
{
DEFINICOES
{
definicao1; definicao2;
}
MOVIMENTOS
instrucao1;
instrucao2;
....
}
```

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 4: Gramática

```
grammar robot;

/*------------------------------------------------------------------
 * PARSER RULES
 *------------------------------------------------------------------*/

robot
@init {
    terreno = new Terreno();
    robo = new Robo(terreno);
}
@after {
    System.out.println(terreno.toString());
    System.out.println(robo.toString());
    System.out.println(robo.toStringEstatisticas());

    Matrix m = new Matrix(robo, terreno);
    m.setVisible(true);
}
    : 'ASPIRADOR' '{' corpo '}'
    ;

corpo
    : 'DEFINICOES' definicoes 'MOVIMENTOS' movimentos
    ;

definicoes
    : '{' dimensao (posicao)? '}'
    | '{' (posicao)? dimensao '}'
    ;
dimensao
    :DIM '=' '(' INT ',' INT ')' ';'
    ;
posicao
    :POS '=' '(' INT ',' INT ')' ';'
    ;

movimentos
    : movimento (movimento)*
    ;

movimento
    : LIGAR ';'
    | DESLIGAR ';'
    | NORTE INT ';'
    | SUL INT ';'
    | ESTE INT ';'
    | OESTE INT ';'
    ;

/*------------------------------------------------------------------
```

```
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 5: Frase gerada 1

```
ASPIRADOR
{
    DEFINICOES
    {
        dim = (100 , 150) ; pos = (0 , 0) ;
    }
    MOVIMENTOS
        LIGAR;
        NORTE 2 ;
        DESLIGAR ;
        SUL 10;
    }
```

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

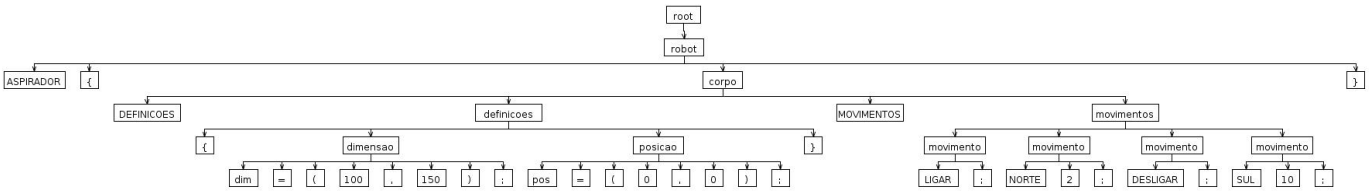

Figura 2: Árvore de derivação

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 dividio em 2 partes: `definicoes` e `movimentos`. Nas `definicoes` podemos configurar a `dimensao` do terreno e ainda a `posicao` inicial do robo.

Quanto aos `movimentos`, este 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.

3.3 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.

3.4 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`.

3.4.1 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.

3.5 Gramática Final

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 6: Gramática Final do Robo

```
grammar robot;

options {
     language = Java;
}

@header{
    import Robot.Robo;
    import Robot.Terreno;
    import Robot.Matrix;
}

@members{
    private Robo robo;
    private Terreno terreno;
}

/*------------------------------------------------------------------
 * PARSER RULES
 *------------------------------------------------------------------*/

robot
@init {
    terreno = new Terreno(); // instancia o terreno
    robo = new Robo(terreno); // instancia o robo
}
@after {
    System.out.println(terreno.toString()); //Imprime o valor da unidade (em cm), a
        largura e altura do terreno
    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
    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

    Matrix m = new Matrix(robo, terreno); // instancia a matrix
    m.setVisible(true);
}
    : 'ASPIRADOR' '{' corpo '}'
    ;

```

```
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 final for valida, entao movimenta o robo para essa posicao
73                 else {System.out.println("Movimento SUL "+ $INT.text +" invalido por ultrapassar os limites da area de limpeza!");}
74               }
75     | ESTE INT ';'      { if (terreno.validaPosX(robo.getPosX() + Integer.parseInt($INT.text))) {robo.movEste(Integer.parseInt($INT.text));} // se a posicao final for valida, entao movimenta o robo para essa posicao
76                 else {System.out.println("Movimento ESTE "+ $INT.text +" invalido por ultrapassar os limites da area de limpeza!");}
77               }
78     | OESTE INT ';'     { if (terreno.validaPosX(robo.getPosX() - Integer.parseInt($INT.text))) {robo.movOeste(Integer.parseInt($INT.text));} // se a posicao
```

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

3.6 Resultado Final

Depois de criada a linguagem, se testarmos com o input:

```
ASPIRADOR
{
    DEFINICOES
    {
    dim = (15 , 15) ; pos = (7 , 7) ;
    }
    MOVIMENTOS
        LIGAR;
        NORTE 2 ;
        ESTE 150 ;
        ESTE 3 ;
        ESTE 2 ;
        SUL 1 ;
        OESTE 5 ;
        SUL 5;
```

```
        DESLIGAR ;
        SUL 0;
        NORTE 10;
        LIGAR;
        OESTE 4;
}
```

vamos obter dois tipos de output, um na consola e outro gráfico.

Output em texto:

```
Movimento ESTE 150 inválido por ultrapassar os limites da área de limpeza!

Terreno{uni=25, larg=15, alt=15}

Robo{posx=3, posy=11, posx_ini=7, posy_ini=7,
     estado=LIGADO, dir=OESTE,
     norte=[2], sul=[1, 5], este=[3, 2], oeste=[5, 4], mud_dir=5,
movs={0=Movimento{num=0, direcao=NORTE, distancia=2},
     1=Movimento{num=1, direcao=ESTE, distancia=3},
     2=Movimento{num=2, direcao=ESTE, distancia=2},
     3=Movimento{num=3, direcao=SUL, distancia=1},
     4=Movimento{num=4, direcao=OESTE, distancia=5},
     5=Movimento{num=5, direcao=SUL, distancia=5},
     6=Movimento{num=6, direcao=OESTE, distancia=4}},
totalMovs=7}

ESTATISTICAS
    Norte:
        Total deslocações: 1
        Total distancia percorrida: 50
        Media de distancia percorrida por cada movimentacao: 50.0
    Sul:
        Total deslocações: 2
        Total distancia percorrida: 150
        Media de distancia percorrida por cada movimentacao: 75.0
    Este:
        Total deslocações: 2
        Total distancia percorrida: 125
        Media de distancia percorrida por cada movimentacao: 62.5
    Oeste:
        Total deslocações: 2
        Total distancia percorrida: 225
        Media de distancia percorrida por cada movimentacao: 112.0
    TOTAL:
        Total deslocações: 7
        Total distancia percorrida: 550
        Media de distancia percorrida por cada movimentacao: 78.57143
        Total mudancas direcao: 5
```

Analisando este output, o que é impresso primeiro é o movimento inválido, depois quando o programa chega ao fim faz o toString das classes `Terreno` e `Robo`, seguidas das estatísticas.

Interface gráfico:

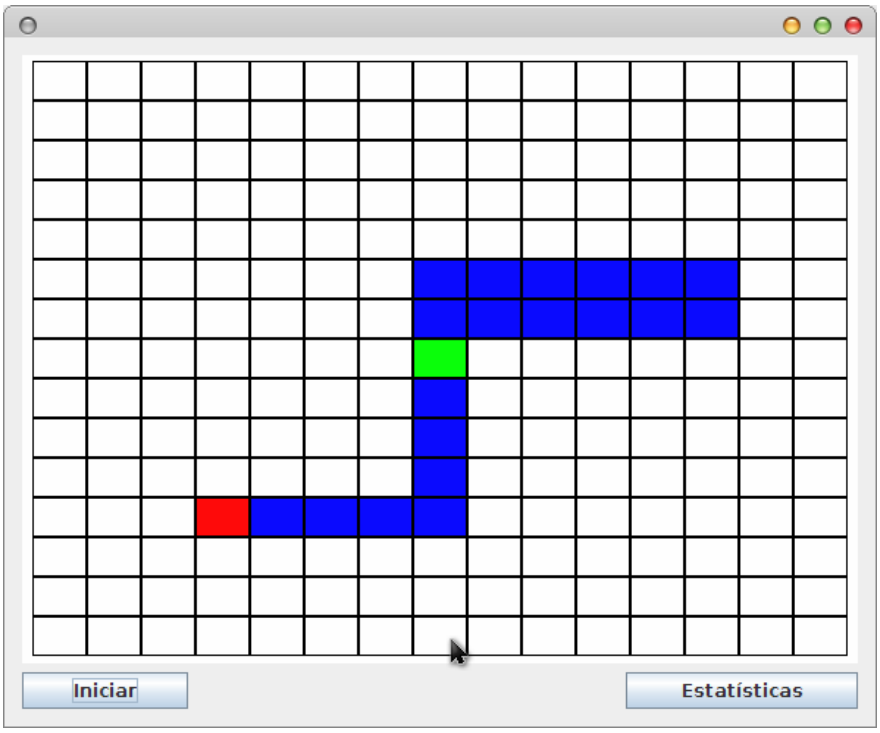

Figura 3: Terreno percorrido pelo Robo

A célula preenchida de cor verde corresponde à posição inicial em que o Robo foi colocado e a célula vermelha à posição final.

Analisando a frase fornecida como input, podemos concluir que as células pintadas corresponde ao trajecto introduzido.

Temos ainda a opção de clicar no botão estatísticas que nos apresenta a seguinte informação:

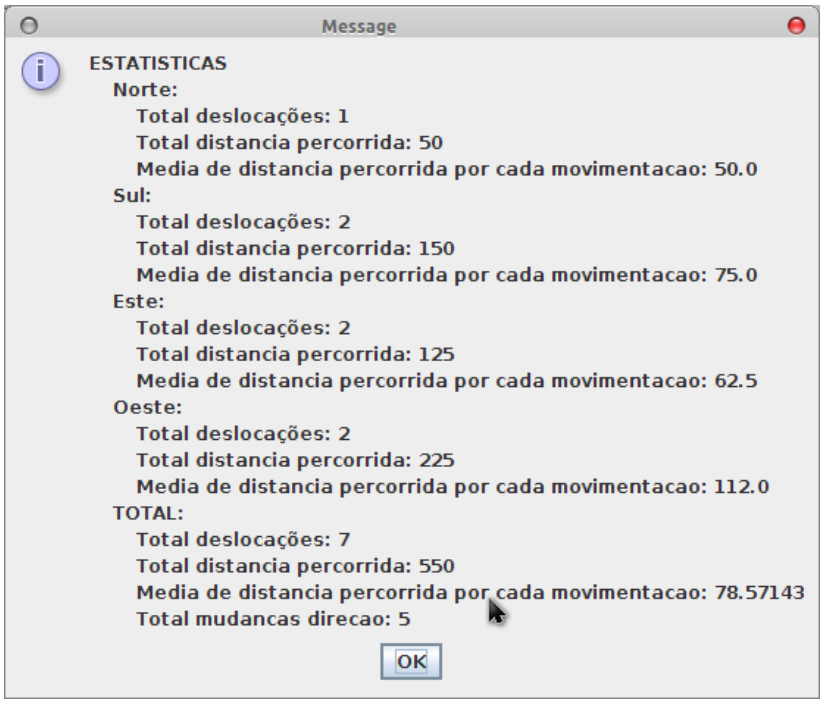

Figura 4: Estatísticas do Robo

4 Conclusões

A resolução deste exercício permitiu perceber melhor a forma como as linguagens podem ser úteis para gerar um programa, que dependo 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.

Umas 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 pertendido.

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

5 Anexos

5.1 Classes em Java

5.1.1 Robo.java

Listing 7: 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
           defeito = 0
38     private int posy_ini = 0;                      // Posicao y inicial por
           defeito = 0
39     private Estado estado;                         // estado do robo (ligado ou
            desligado)
40     private Direcao dir;                           // direcao atual do
           aspirador
41     private ArrayList<Integer> norte = new ArrayList<Integer>();  // array
           que armazena os movimentos na direcao norte
42     private ArrayList<Integer> sul = new ArrayList<Integer>();    // array
           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;                    // aramazena 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 referÃªncia 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 Devolveo 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
           correspondente
172      * @param num Numero do movimento
173      * @return Classe Movimento que corresponde ao numero dado como
           parametro
174      */
175     public Movimento getMovimento(int num) {
176         return movs.get(num).clone();
177     }
178 
179     /**
180      * Altera a Posicao X do robo.
181      * Usada quando e preciso definiar a Posicao inicial e quando a
           movimentacao
182      * @param x Nova posicao x em que o robo vai ficar
183      */
184     public void setPosX(int x) {
185         posx = x;
186     }
187 
188     /**
189      * Altera a Posicao X do robo recebendo uma string que contem o valor
           inteiro
190      * Usada quando e preciso definiar a Posicao inicial e quando a
           movimentacao
191      * @param x Nova posicao x em que o robo vai ficar
192      */
193     public void setPosX(String x) {
194         posx = Integer.parseInt(x);
195     }
196 
197     /**
198      * Altera a Posicao Y do robo
199      * Usada quando e preciso definir a Posicao inicial e quando ha
           movimentacao
200      * @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
               inteiro,
208         * usada quando e preciso definir a Posicao inicial e quando ha
               movimentacao
209         * @param y Nova posicao y em que o robo vai ficar
210         */
211        public void setPosY(String y) {
212            posy = Integer.parseInt(y);
213        }
214
215        /**
216         * Altera a Posicao X do robo.
217         * Usada quando e preciso definiar a Posicao inicial
218         * @param x Nova posicao x em que o robo tem inicialmente
219         */
220        public void setPosXini(int x) {
221            posx_ini = x;
222        }
223
224        /**
225         * Altera a Posicao X do robo recebendo uma string que contem o valor
               inteiro
226         * Usada quando e preciso definiar a Posicao inicial
227         * @param x Nova posicao x em que o robo tem inicialmente
228         */
229        public void setPosXini(String x) {
230            posx_ini = Integer.parseInt(x);
231        }
232
233        /**
234         * Altera a Posicao Y do robo
235         * Usada quando e preciso definir a Posicao inicial
236         * @param y Nova posicao y em que o robo tem inicialmente
237         */
238        public void setPosYini(int y) {
239            posy_ini = y;
240        }
241
242        /**
243         * Altera a Posicao Y do robo recebendo uma string que contem o valor
               inteiro
244         * Usada quando e preciso definir a Posicao inicial
245         * @param y Nova posicao y em que o robo tem inicialmente
246         */
247        public void setPosYini(String y) {
248            posy_ini = Integer.parseInt(y);
249        }
250
251        /**
252         * 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
                   igual
318            // a zero, ignora e apenas muda a direcao para norte
319            if (dist > 0) {
320                // contabiliza ou nao a mudanca de direccao
321                if (dir != Direcao.NORTE && dir != Direcao.NULA) {
322                    mud_dir++;
323                }
324                // altera o Posicao y
325                posy -= dist;
326                // adiciona a distancia ao array de movimentacoes para norte
327                norte.add(dist);
328                // regista no TreeMap o movimento
329                Movimento m = new Movimento(totalMovs, Movimento.Direcao.NORTE,
                       dist);
330                movs.put(totalMovs, m);
331                totalMovs++;
332            }
333            // define a direcao para norte
334            dir = Direcao.NORTE;
335        }
336
337        /**
338         * Move o robo uma certa distancia para Sul
339         * @param dist
340         */
341        public void movSul(int dist) {
342            // se estiver desligado nao faz nada
343            if (estado == Estado.DESLIGADO) {
344                return;
345            }
346
347            // se a distancia for maior que zero faz coisas, se for menos ou
                   igual
348            // a zero, ignora e apenas muda a direcao para sul
349            if (dist > 0) {
350                // contabiliza ou nao a mudanca de direccao
351                if (dir != Direcao.SUL && dir != Direcao.NULA) {
352                    mud_dir++;
353                }
354                // altera o Posicao y
355                posy += dist;
356                // adiciona a distancia ao array de movimentacoes para sul
357                sul.add(dist);
358                // regista no TreeMap o movimento
359                Movimento m = new Movimento(totalMovs, Movimento.Direcao.SUL,
                       dist);
360                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
                igual
378         // a zero, ignora e apenas muda a direcao para este
379         if (dist > 0) {
380             // contabiliza ou nao a mudanca de direccao
381             if (dir != Direcao.ESTE && dir != Direcao.NULA) {
382                 mud_dir++;
383             }
384             // altera o Posicao x
385             posx += dist;
386             // adiciona a distancia ao array de movimentacoes para este
387             este.add(dist);
388             // regista no TreeMap o movimento
389             Movimento m = new Movimento(totalMovs, Movimento.Direcao.ESTE,
                    dist);
390             movs.put(totalMovs, m);
391             totalMovs++;
392         }
393         // define a direcao para sul
394         dir = Direcao.ESTE;
395     }
396 
397     /**
398      * Move o robo para Oeste
399      * @param dist
400      */
401     public void movOeste(int dist) {
402         // se estiver desligado nao faz nada
403         if (estado == Estado.DESLIGADO) {
404             return;
405         }
406 
407         // se a distancia for maior que zero faz coisas, se for menos ou
                igual
408         // a zero, ignora e apenas muda a direcao para oeste
409         if (dist > 0) {
410             // contabiliza ou nao a mudanca de direccao
411             if (dir != Direcao.OESTE && dir != Direcao.NULA) {
412                 mud_dir++;
413             }
```

```
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 }
```

5.1.2 Terreno.java

Listing 8: 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    }
```

5.1.3 Movimento.java

Listing 9: Movimento.java

```
package Robot;

/**
 * Classe que guarda o registo de um movimento
 * @author Bruno Azevedo, Miguel Costa
 */
public class Movimento {

    public enum Direcao {

        NULA,
        NORTE,
        SUL,
        ESTE,
        OESTE
    }
    private Integer num;
    private Direcao direcao;
    private Integer distancia;

    public Movimento(Integer n, Direcao d, Integer dist) {
        num = n;
        direcao = d;
        distancia = dist;
    }

    private Movimento(Movimento m) {
        num = m.getNum();
        direcao = m.getDirecao();
        distancia = m.getDistancia();
    }

    public Direcao getDirecao() {
        return direcao;
    }

    public Integer getDistancia() {
        return distancia;
    }

    public Integer getNum() {
        return num;
    }

    @Override
    public Movimento clone() {
        return new Movimento(this);
    }

    @Override
    public String toString() {
        return "Movimento{" + "num=" + num + ", direcao=" + direcao + ",
            distancia=" + distancia + '}';
```

```
53          }
54  }
```

5.1.4 Matrix.java

Listing 10: 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).addGap(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             this.update(this.getGraphics());
131             //this.update(paneis[posicao(x, y)].getGraphics());
132             try {
133                 Thread.sleep(500);
134             } catch (InterruptedException ex) {
135                 Logger.getLogger(Matrix.class.getName()).log(Level.SEVERE, null,
                        ex);
136             }
137         }
138 
139         /**
140          * Pinta a celula correspondente a posicao inicial do robo
141          */
142         private void pinta_ini() {
143             int x = _r.getPosXini();
144             int y = _r.getPosYini();
145             paneis[posicao(x, y)].setBackground(new java.awt.Color(10, 254, 10))
                    ;
146             this.update(this.getGraphics());
147             try {
148                 Thread.sleep(500);
149             } catch (InterruptedException ex) {
150                 Logger.getLogger(Matrix.class.getName()).log(Level.SEVERE, null,
                        ex);
151             }
152         }
153 
154         /**
155          * 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            this.update(this.getGraphics());
162            try {
163                Thread.sleep(500);
164            } catch (InterruptedException ex) {
165                Logger.getLogger(Matrix.class.getName()).log(Level.SEVERE, null,
                       ex);
166            }
167        }
168
169        /** This method is called from within the constructor to
170         * initialize the form.
171         * WARNING: Do NOT modify this code. The content of this method is
172         * always regenerated by the Form Editor.
173         */
174        @SuppressWarnings("unchecked")
175        // <editor-fold defaultstate="collapsed" desc="Generated Code">//GEN-
               BEGIN:initComponents
176        private void initComponents() {
177
178            jPanelMatrix = new javax.swing.JPanel();
179            jButtonStart = new javax.swing.JButton();
180            jButtonStats = new javax.swing.JButton();
181
182            setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
183
184            jPanelMatrix.setBackground(new java.awt.Color(254, 254, 254));
185            jPanelMatrix.setLayout(new java.awt.GridLayout(3, 3));
186
187            jButtonStart.setText("Iniciar");
188            jButtonStart.addActionListener(new java.awt.event.ActionListener() {
189                public void actionPerformed(java.awt.event.ActionEvent evt) {
190                    jButtonStartActionPerformed(evt);
191                }
192            });
193
194            jButtonStats.setText("EstatÃsticas");
195            jButtonStats.addActionListener(new java.awt.event.ActionListener() {
196                public void actionPerformed(java.awt.event.ActionEvent evt) {
197                    jButtonStatsActionPerformed(evt);
198                }
199            });
200
201            javax.swing.GroupLayout layout = new javax.swing.GroupLayout(
                   getContentPane());
202            getContentPane().setLayout(layout);
203            layout.setHorizontalGroup(
204                layout.createParallelGroup(javax.swing.GroupLayout.Alignment.
                       LEADING)
205                .addGroup(layout.createSequentialGroup()
206                    .addContainerGap()
```

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

```
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 }
```