

# PROJETO LFA

Linguagem para questionários interativos

QUIZ GENERATOR (QG)

Departamento Eletrónica, Telecomunicações e Informática



universidade  
de aveiro

## Autores

*Alexey Kononov - 89227*

*Diogo Fernandes - 89221*

*Henrique da Silva - 88857*

*Martim Neves - 88904*

*Tiago Dias - 88896*

## Índice

Objetivos .....	3
Manual de Instruções .....	4
Palavras Reservadas .....	5
Read Questions.....	6
Quiz Generator.....	7
Criação de um questionário.....	10
Semantic Check (exemplos).....	12
Figura 1 .....	12
Figura 2.....	12
Figura 3.....	13
Figura 4.....	13
Contribuição dos Autores .....	14
Conclusão e Resultados .....	15

## Objetivos

O principal objetivo deste projeto passa por desenvolver uma linguagem que permita ao utilizador a geração de um questionário com questões de dificuldade e temas à sua escolha.

Essas questões encontram-se numa base de dados, divididas por tema e nível de dificuldade, onde cada pergunta tem quatro opções de resposta, sendo que apenas uma está correta.

Foram definidas duas gramáticas, a ReadQuestions e a QuizGenerator. A ReadQuestions permite ao programa ler a base de dados onde estão as questões. Já a QuizGenerator tem como função gerar o questionário.

Foi também criado um sistema de pontuação em que cada resposta certa dá 100 pontos e cada errada tira 10% da pontuação atual.

A linguagem escolhida para embutir o questionário foi Java.

## Manual de Instruções

- Começar por executar o comando **./build.sh** para compilar todos os ficheiros;
- Todos os ficheiros de teste estão disponíveis no diretório *Test*;
- De seguida correr o comando **java QuizGeneratorMain < Test/testGeneratedCode.QG** que permite testar o código que queremos usar para gerar o questionário, passando pelo semanticCheck, e ao mesmo tempo compilá-lo para o programa java que vai correr o questionário sendo o código gerado impresso no terminal,;
- O comando **java QuizGeneratorMain -f < Test/testGeneratedCode.QG** exporta o código gerado para um ficheiro java;
- O comando **java QuizGeneratorMain -s < Test/testGeneratedCode.QG** imprime no terminal as mensagens do semanticCheck e o código gerado;
- O comando **java QuizGeneratorMain -f-s < Test/testGeneratedCode.QG** exporta o código gerado para um ficheiro java e exporta as mensagens do semanticCheck para um ficheiro .txt;
- Para testar o questionário gerado, é necessário compilá-lo com o comando **javac testQuest.java**;
- Por fim corremos o programa com o comando **java testQuest**.

## Palavras Reservadas

Tal como em todas as linguagens de Programação, também a QG tem um conjunto de palavras reservadas que não podem ser usadas para atribuir nomes a variáveis, métodos, etc.

begin	create	endcreate	print
for	endfor	if	endif
rand	question	string	int
double	DB	answersMode	load
userAnswer			

Para além das palavras mais comuns em muitas linguagens (for, if, rand, ...), existem palavras menos habituais (answersMode, endcreate, ...) que serão explicadas mais à frente.

## Read Questions

A gramática ReadQuestions tem como propósito a leitura da base de dados onde se encontram as question que vão estar no questionário.

Uma question é composta por:

- QuestionID - que por sua vez é composto por:
  - ID(tema).ID(numero da pergunta).Difficulty;
- Seguido de uma String – a pergunta em si;
- E por fim por uma ou mais answers, que correspondem às hipóteses de resposta, e onde cada resposta tem o seguinte formato:
  - String(resposta) : Number(pontuação);

```
soccer.p1.easy("Qual o acrónimo mais conhecido de Cristiano Ronaldo?"){  
    "CR17" : 0;  
    "CR28" : 0;  
    "CR7" : 100;  
    "R9" : 0;  
}
```

Acima é mostrado um exemplo de uma question, onde o QuestionID é representado por – soccer(tema).p1(numero da pergunta).easy(dificuldade da pergunta).

A String que corresponde à pergunta em si está entre parenteses e, por fim, as answers (4 neste caso) estão entre chavetas. São compostas por uma String, que corresponde à resposta e pelo Number que equivale à pontuação.

É também de destacar que esta gramática ignora os comentários feitos no programa java, permitindo assim a escrita dos mesmos.

## Quiz Generator

A gramática QuizGenerator tem como propósito a geração de um questionário.

Tal como na gramática ReadQuestions, também na QuizGenerator é permitida a escrita de comentários, pois estes são ignorados para efeitos de execução do código.

A criação de um questionário é começada por um `Begin create ID` e terminada com um `endcreate`.

Pelo meio podem ser definidos `stat's`. Um `stat` pode ser uma instrução (`instruction`), um bloco `for` (`forBlock`) ou um bloco `if` (`ifBlock`).

- **instruction** – pode ser um `assignment`, um `command` ou um `createQuestion`.
  - Um **assignment** por sua vez pode ser uma `declaration`, `attribution`, `questionDeclaration`, `questionAttribution` ou `bdAttribution`.
    - **declaration** permite declarar variáveis (`type ID`) ou arrays (`type '[]' ID`).
    - **attribution** tal como o nome indica serve para atribuir valores às variáveis e arrays declarados.
    - **questionDeclaration** permite declarar uma variável do tipo `question` (`questionType ID`) ou um array de `questions` (`questionType '[]' ID`).
    - **questionAttribution** possibilita a atribuição de valores às `questions` e aos arrays de `questions` criados.
    - **bdAttribution** consiste em carregar a base de dados com as questões que farão parte do questionário.
  - Um **command** permite fazer uma série de operações.
    - Escolher o tipo de teste – `testType`, que pode ser *multipleChoice* ou *trueOrFalse*. Ex: `answersMode = multipleChoice`
    - Adicionar perguntas ao questionário. Ex: `questions.add(quest);`

- ‘Baralhar’ as possibilidades de resposta. Ex: *rand question.answers()*;
  - Escolher o número de respostas por pergunta. Ex: *question.numAnswers(4)*;
  - Guardar a resposta do user numa var. Ex: *choice = userAnswer*;
- **forBlock** – consiste numa palavra reservada ‘for’ seguida de um ID, de outra palavra reservada ‘in’ e de outro ID. Dentro do ciclo podem ser definidos stat’s. É terminado com um ‘endfor’.

```
for question in questions:

    questions.print(question);

endfor;
```

- **ifBlock** – consiste numa palavra reservada ‘if’ seguida de uma condition. Pode ter stat’s e um other(else). É terminado com um ‘endif’.
- Uma **condition** consiste numa comparação. Por exemplo – *mathExpr ‘==’ mathExpr*, *mathExpr ‘>’ mathExpr*.
  - **mathExpr** é um tipo de expr. Pode ser um numero, um ID(como no exemplo abaixo), ou uma operação de *mathExpr*’s – *mathExpr op mathExpr*, sendo que op = + | - | \* | / .

```
If (choice == question.correctAnswer() ):

    Score = score + 100;

else:

    Score = score – (score * 0.1);

endif;
```



- **createQuestion** – o utilizador tem também a possibilidade de gerar ele mesmo uma questão. Para tal existe o createQuestion (enquadra-se dentro da instruction). Deverão ser especificados alguns campos da question, como o seu tema e a pergunta em si, entre outros, conforme demonstrado no exemplo abaixo.

<i>Question newQuest.text = "Quantas patas tem um gato?"</i>	<i>//Declaração da nova pergunta e do seu texto</i>
<i>newQuest.theme = "animals"</i>	<i>//Tema da pergunta</i>
<i>newQuest.name = "P7"</i>	<i>//Nome (identificador) da pergunta</i>
<i>newQuest.difficulty = easy</i>	<i>//Dificuldade da pergunta</i>
<i>newQuest.answer("4 patas", 100)</i>	<i>//Criação de várias hipóteses de</i>
<i>newQuest.answer("2 patas", 0)</i>	<i>//resposta e da pontuação</i>
<i>newQuest.answer("3 patas", 0)</i>	<i>//correspondente, isto é, se</i>
<i>newQuest.answer("1 patas", 0)</i>	<i>//está certa ou errada</i>
<i>questions.add(newQuest)</i>	<i>//Adição ao array de perguntas</i>

## Criação de um questionário

O código gerado a seguir, cria um teste com 3 perguntas do tema animais, 3 perguntas de história, 3 perguntas de matemática e 1 sobre línguas. As respostas são do tipo escolha múltipla, cada uma com 4 opções de resposta.

```
Begin create quest1                                // cria o questionário chamado 'quest1'

Question[ ] questions;
String choice;
double score = 0;
String[] themes = ["animals", "history",
"maths"];
Question newQuest.text = "Quantas patas
tem um gato?";

newQuest.theme = "animals";                        // facultativo pode ser null por default

newQuest.name = "P7";                              // dá o nome à pergunta

newQuest.difficulty = easy;                        // dificuldade da pergunta

newQuest.answer("4 patas", 100);
newQuest.answer("2 patas", 0);                    //adiciona esta opção às hipóteses de resposta
newQuest.answer("3 patas", 0);
newQuest.answer("1 patas", 0);

questions.add(newQuest);                          //adiciona a questão criada ao array de questões

answersMode = multipleChoice;                      // Define quiz com respostas de escolha múltipla

DB data = load("bd1.question");                    // carrega a base de dados com o ficheiro das
perguntas

data.add(newQuest);                                // Acrescenta a questão criada à base de dados

for theme in themes:                               //percorre os temas existentes no array de temas

    Question [] quest = data.get(3, medium,        // obtém 3 perguntas com dificuldade média sobre
theme);                                             o tema 'theme'
```

```

questions.add(question);           // adiciona as perguntas anteriormente obtidas ao
                                   // array à lista 'questions' inicialmente definida

endfor                             // acaba o for

Question simpleQuest = data.get(easy, // obtém 1 pergunta com dificuldade fácil sobre o
"languages");                      // tema 'Languages'

questions.add(simpleQuest);         // adiciona a pergunta anteriormente obtida às
                                   // 'questions'

rand questions;                    // coloca as perguntas de forma aleatória

for question in questions:

    question.numAnswers(4);         //cada pergunta tem 4 escolhas

    rand question.answers();         // coloca a ordem das respostas de forma aleatória

    print question;                 // faz print da pergunta 'question'

    choice = userAnswer;             // choice é a resposta do utilizador
                                   // se o utilizador acertar na resposta

    if (choice == question.correctAnswer()): // question.correctAnswer() -> devolve a resposta
                                   // correta da pergunta 'question'

        score = score + 100;         // aumenta a pontuação

    else:                            // caso o utilizador falhe

        score = score - (score * 0.1); // retira 10% á pontuação

    endif                            // acaba o if

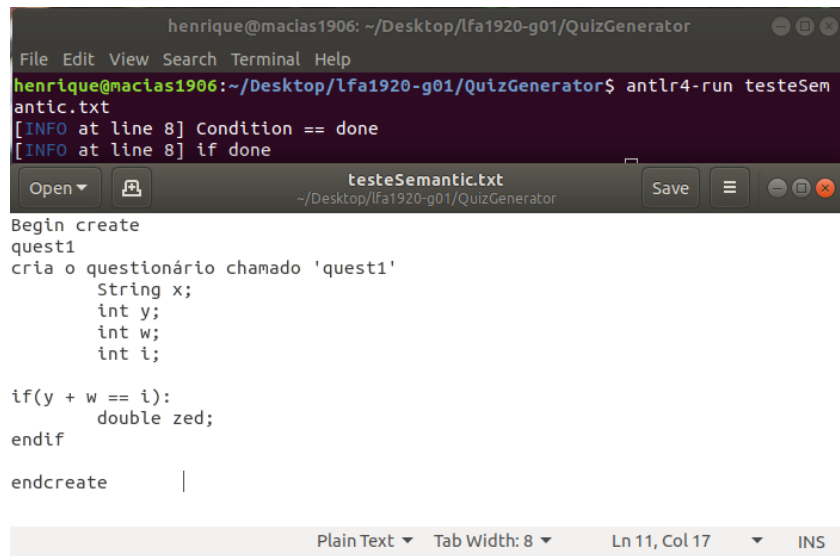
endfor                              // imprime o score

endcreate                           // acaba criação do questionário 'quest1'

```

## Semantic Check (exemplos)

Na Figura 1 o código está semanticamente correto, portanto o semantic Check deixa passar sem dar erros. No entanto, na Figura 2 é comparada uma String com um Double sendo assinalado o erro.



The screenshot shows a terminal window at the top with the command `antlr4-run testeSemantic.txt` and two informational messages: `[INFO at line 8] Condition == done` and `[INFO at line 8] if done`. Below the terminal is a code editor window titled `testeSemantic.txt` containing the following code:

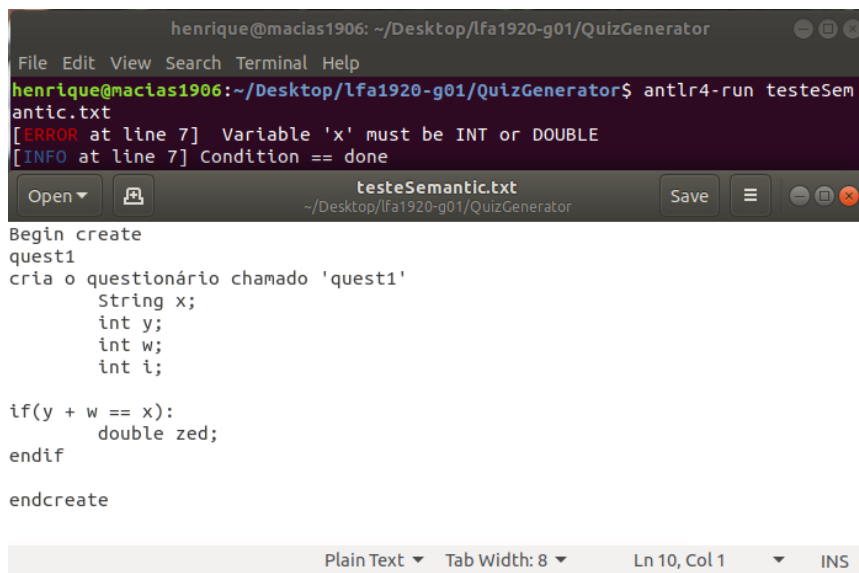
```
Begin create
quest1
cria o questionário chamado 'quest1'
    String x;
    int y;
    int w;
    int i;

if(y + w == i):
    double zed;
endif

endcreate
```

The status bar at the bottom indicates 'Plain Text', 'Tab Width: 8', 'Ln 11, Col 17', and 'INS'.

Figura 1



The screenshot shows a terminal window at the top with the command `antlr4-run testeSemantic.txt` and two messages: an error `[ERROR at line 7] Variable 'x' must be INT or DOUBLE` and an informational message `[INFO at line 7] Condition == done`. Below the terminal is a code editor window titled `testeSemantic.txt` containing the following code:

```
Begin create
quest1
cria o questionário chamado 'quest1'
    String x;
    int y;
    int w;
    int i;

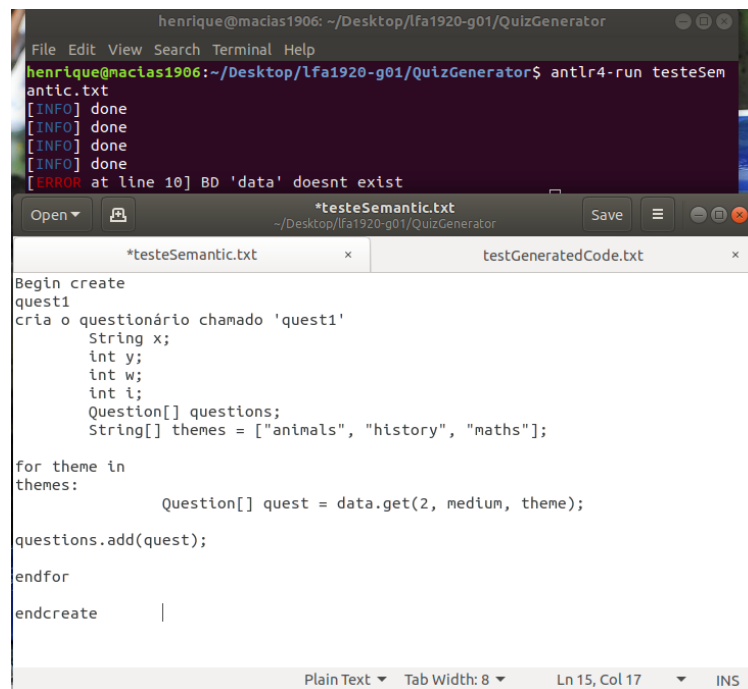
if(y + w == x):
    double zed;
endif

endcreate
```

The status bar at the bottom indicates 'Plain Text', 'Tab Width: 8', 'Ln 10, Col 1', and 'INS'.

Figura 2

Nas Figuras 3 e 4, é parcialmente criado um objeto do tipo Question que posteriormente é acrescentada à Base de Dados. No entanto, na Figura 3 a DB não é instanciada antes desta ação, logo o Semantic Check aponta o erro. Já na Figura 4 o programa corre sem problemas pois o objeto do tipo DB é instanciado.



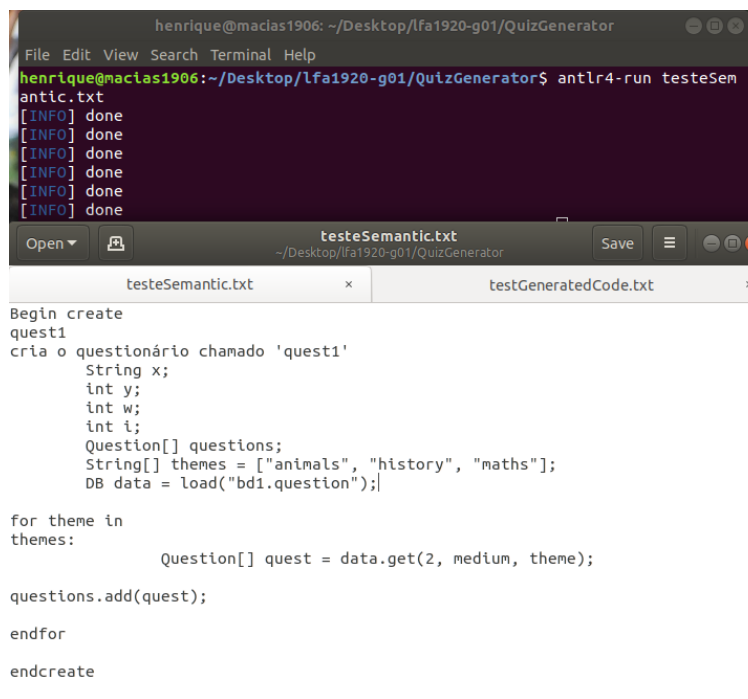
The screenshot shows a terminal window at the top with the command `antlr4-run testeSemantic.txt` and its output: `[INFO] done` repeated five times, followed by `[ERROR] at line 10] BD 'data' doesnt exist`. Below the terminal is a code editor with two tabs: `*testeSemantic.txt` and `testGeneratedCode.txt`. The `*testeSemantic.txt` tab is active and contains the following code:

```
Begin create
quest1
cria o questionário chamado 'quest1'
    String x;
    int y;
    int w;
    int i;
    Question[] questions;
    String[] themes = ["animals", "history", "maths"];

for theme in
themes:
    Question[] quest = data.get(2, medium, theme);
questions.add(quest);
endfor
endcreate
```

The status bar at the bottom indicates 'Plain Text', 'Tab Width: 8', 'Ln 15, Col 17', and 'INS'.

Figura 3



The screenshot shows a terminal window at the top with the same command `antlr4-run testeSemantic.txt` and output, but the output now consists of `[INFO] done` repeated six times, with no error. Below the terminal is the same code editor with the same tabs. The `*testeSemantic.txt` tab is active and contains the following code, which is identical to Figure 3 but includes an additional line: `DB data = load("bd1.question");`Begin create
quest1
cria o questionário chamado 'quest1'
 String x;
 int y;
 int w;
 int i;
 Question[] questions;
 String[] themes = ["animals", "history", "maths"];
 DB data = load("bd1.question");

for theme in
themes:
 Question[] quest = data.get(2, medium, theme);
questions.add(quest);
endfor
endcreate

The status bar at the bottom indicates 'Plain Text', 'Tab Width: 8', 'Ln 15, Col 17', and 'INS'.

Figura 4

## Contribuição dos Autores

- Alexey Kononov – 20%
  - String Template
  - Semantic Check
  - QuizGenerator
  - Tradução de Código Fonte
- Diogo Fernandes – 20%
  - QuizGenerator
  - Semantic Check
  - Perguntas para a Base de Dados
  - ReadQuestion
- Henrique Silva – 20%
  - QuizGenerator
  - Semantic Check
  - Perguntas para a Base de Dados
  - Relatório
- Martim Neves – 20%
  - QuizGenerator
  - Semantic Check
  - Tradução de Código Fonte
  - Relatório
- Tiago Dias – 20%
  - String Template
  - Tradução de Código Fonte
  - ReadQuestion
  - Compiler

## Conclusão e Resultados

Através dos conhecimentos adquiridos ao longo do semestre nas aulas teóricas e práticas e, também, de reuniões e sessões para esclarecimento de dúvidas, foi-nos possível ter o alicerce necessário para a realização deste projeto da cadeira de LFA.

No geral, todos os objetivos e desafios propostos foram concretizados e ultrapassados e a finalidade do projeto, isto é, a criação de um questionário através de uma linguagem desenvolvida por nós, foi atingida.

Por fim, o nosso agradecimento ao professor Artur pela constante disponibilidade em ajudar e esclarecer dúvidas.