Universidade de Aveiro

Projeto nº 04
Linguagem para Questionários Interativos

Alexandre Tomás (93289), Ana Filipe (93350), Gabriel Ribeiro (93036),

João Gameiro (93097), Pedro Abreu (93240)

Linguagens Formais e Autómatos

Departamento de Eletrónica, Telecomunicações e Informática

17 de Junho de 2020

Conteúdo

Lista de Tabelas

Lista de Figuras

Capítulo 1

Introdução

Um questionário é uma ferramenta de elevada importância nos dias de hoje, que pode auxiliar a recolha informação ou servir de modelo de teste para avaliação de conhecimentos, tendo múltiplas utilizações.
Deste modo, surge ***Kuest***, uma linguagem bastante simples que apresenta várias semelhanças com outras linguagens de mais alto nível e ferramentas adicionais para desenvolvimento de questionários interativos.
Kuest apresenta possibilidade de definição de variáveis de diferentes tipos, assim como utilização de ciclos e instruções condicionais. Contém também instruções para a apresentação e modulação de questionários, assim como a possibilidade de processar respostas dadas e exportação das mesmas para a sua posterior análise. A linguagem destino de ***Kuest*** é ***Java***, ou seja quando compilamos um ficheiro de ***Kuest*** vai ser originado um ficheiro ***Java***.
Tendo uma linguagem para o desenvolvimento de questionários, é também importante a existência de uma base de dados que permita guardar perguntas para a sua posterior utilização nos questionários.
Logo surge neste contexto outra linguagem bastante mais simples que permite ao programador descrever ficheiros de perguntas que são carregadas para uma base de dados que pode ser utilizada ou não para o desenvolvimento de questionários.
Para a melhor perceção de ambas as linguagens foi desenvolvido este relatório no qual irão ser descritas as linguagens, apresentados exemplos de utilização das mesmas e um manual de instruções de como compilar e correr os programas desenvolvidos.

Capítulo 2

Linguagem para Base de Dados

Para que o utilizador possa usufruir de uma base de dados de perguntas, foi criada uma linguagem simples, que tem como objetivo a descrição de ficheiros que contém perguntas. Após a compilação desse tipo de ficheiros, todas as perguntas descritas são carregadas para uma base de dados que poderá ser posteriormente utilizada na linguagem ***Kuest***.

```
//sintaxe de uma pergunta
tema-ID-dificuldade["Pergunta"]{
    "resposta 1" - cotacao1;
    "resposta 2" - cotacao2;
    "resposta 3" - cotacao3;
}
```

Tabela 2.1: Constituintes de uma pergunta

tema	identificador(pode ser descrito com letras e números)
ID	identificador(pode ser descrito com letras e números)
dificuldade	apenas pode ser descrita por números (1 a 5)
pergunta	string(texto limitado por aspas)
resposta	string(resposta) e um inteiro cotação(0 a 10)

Assim é fácil de perceber que a base de dados será organizada num Mapa de temas em que cada chave será representada por uma string que identifica o tema, à qual corresponde uma Lista de questões.
É importante referir que dentro de cada tema, não são permitidos: ids iguais nem perguntas iguais. Também não são permitidas resposta iguais dentro de uma pergunta e a dificuldade tem de ser um valor entre 0 e 5 (muito fácil a muito difícil).

```
//exemplo de uma pergunta do tema animais, com id = 'p01' e dificuldade 5
animais-p01-5["Qual o maior felino do mundo, em dimensões?"]{
    "Leão" - 0;         //resposta 'Leão' com cotação 0
    "Tigre" - 100;      //resposta 'Tigre' com cotação 100
    "Onça" - 0;
}
```

Se surgir algum erro no programa do utilizador, este será notificado do mesmo.
Só são permitidos comentários de linha nesta linguagem que são identificados pelos caracteres '//'.

Capítulo 3

Linguagem para Desenvolvimento de Questionários

3.1 Instruções

Em ***Kuest*** podem existir instruções de bloco ou instruções de linha. As instruções de linha são terminadas pelo caractere ';'. As instruções de bloco não possuem essa restrição.

```
string a;       #Declaração (Instrução de linha)
display(k);     #Utilização do método display (Instrução de linha)

#Instrução de bloco
if(a = "Hello"){
    i++;
}
```

3.1.1 Declarações e Atribuições

Para declararmos uma variável simplesmente temos de especificar o tipo da variável e a sua designação.
O tipo de uma variável não pode ser alterado durante o restante tempo de vida da mesma.
Se o utilizador tentar usar uma variável que não foi inicializada irá ser notificado do errro.

```
string a;       #Variável "a" do tipo string
array int ai;   #Variável "ai" do tipo array de inteiro
```

Para atribuirmos um valor a uma variável usamos o símbolo '->' (após indicar qual a variável pretendida) e especificamos o seu valor.

```
string a -> "Hello";        #Atribuição do valor "Hello" à variável "a"
int num;                    #Criação da variável num
num -> 8;                   #Atribuição do valor 8 à variável num
```

As variáveis do tipo inteiro permitem também duas operações adicionais identificadas por '++' e '– –' que servem para incrementar ou decrementar a respetiva variável.

```
num++;  #Variável "num" toma o valor de num + 1
num--;  #Variável "num" toma o valor de num - 1
```

3.1.2 Tipos

A linguagem ***Kuest*** apresenta os seguintes tipos

int

Representa um número inteiro, equivalente ao int da linguagem destino.

string

Forma de representação de texto, delimitada por aspas, equivalente ao tipo String da linguagem destino.

boolean

Representação de valores booleanos ***true*** e ***false***.

kuest

Representação das questões. Presume-se que no contexto da linguagem uma questão é caraterizado por um quíntuplo constituído por:

- Tema (representado por uma String);
- Dificuldade (representada por um inteiro);
- ID (caraterizado por uma String);
- Conjunto de respostas (caraterizado por um mapa em que as chaves são do tipo string(resposta) e os valores são inteiros (cotação da resposta));
- Pergunta (representada por uma string).

As questões possuem um método ***get*** que permite ao utilizador obter informação relativa à questão.

```
questao.get(ID);     #Devolve uma string que é o ID da questão
questao.get(theme); #Devolve uma string que é o Tema da questão
questao.get(numberOfanswers); #Devolve o número de respostas de uma questão
questao.get(dificulty)  #Devolve um inteiro que representa a dificuldade da questão
```

É possível ir buscar questões à base de dados ou criá-las. Se quisermos criar uma questão usamos os caracteres '/*' e '*/' como delimitadores. Dentro dos delimitadores escrevemos as características da questão usando a sintaxe da linguagem de descrição de questões.

```
#Criação de uma questão
kuest k -> /* animais-p02-2["Qual o maior réptil do mundo?"]{
    "Lagarto" - 0;
    "Crocodilo" - 0;
    "Tartaruga" - 100;
}*/
```

array

É possível criar arrays de qualquer um dos tipos referidos anteriormente. Na criação de arrays não é necessário especificar o tamanho dos mesmos. Pode-se explicitar os seus valores em extensão ou então adicioná-los posteriormente com o método add.

```
array string a;          #Declaração de um array "a" do tipo string
array int ai -> {1, 2};  #Declaração e inicialização de um array do tipo int
```

Os arrays apresentam uma série de métodos que permitem ao utilizador a manipulação da estrutura de dados:

```
a.add("Hello", "World");  #Método add serve para adicionar elementos ao fim do array
a.remove("Hello");        #Método remove serve para remover elementos do array
#De notar que add e remove permitem ambos variáveis do tipo do array
a.length                  #lenght devolve o número de elementos presente no array
a.shuffle                 #Método shuffle baralha o conteúdo do array aleatoriamente
a.clear                   #Limpa o conteúdo do array
a[0]                      #Aceder ao conteúdo da posição 0 do array
#Importante referir que apenas é permitido o acesso a uma posição isolada, não é
#  permitido alterar o conteúdo de uma posição do array com esta notação
```

3.2 Comentários

Na linguagem ***Kuest*** apenas são permitidos comentários de linha que são inicializados pelo caractere '#'.

```
#Comentário de linha
array int ai;   #Outro comentário de linha
```

3.3 Palavras Reservadas

Kuest possui 29 palavras reservadas (Tabela 1). Nenhuma dessas palavras pode ser utilizado para nome de variável.

3.4 Operações Aritméticas

Todas as operações aritméticas permitidas, e os respetivos tipos que as suportam estão sintetizados na Tabela 3.

Tabela 3.1: Palavras Reservadas

int	boolean	string	kuest	array
true	false	and	or	OR
choose	from	memory	ID	theme
numberOfanswers	dificulty	kuestover	if	elseif
else	for	while	do	in
break	mc	tf	txt	

Tabela 3.2: Operações Aritméticas, suas funções e tipos permitidos

Operação	Função	Tipo
+	Adição ou Sinal Positivo	int
−	Subtração ou Sinal Negativo	int
\	Divisão	int
%	Resto da Divisão inteira	int
*	Multiplicação	int
=	Igualdade	int, boolean, string e kuest
!=	Diferença	int, boolean, string e kuest
<	Menor	int
>	Maior	int
<=	Menor ou Igual	int
>=	Maior ou Igual	int

As palavras reservadas ***and*** e ***or*** são para ser usadas em estruturas condicionais (tal como os operadores de comparação) e representam respetivamente a conjunção e disjunção lógica. Existe também um método ***random(int, int)*** que permite gerar um número inteiro aleatório entre dois valores. Esses dois valores são os argumentos do método.

```
int i -> (2*1) + 4 -- 5;              #Utilização de uma expressão aritmética
if( (i<0 and i>25) or i=10){...};     #Utilização de and e or
i -> random(1, 5);                    #Gerar um número aleatório entre 1 e 5
```

3.5 Instruções Condicionais

A linguagem ***Kuest*** permite instruções que alterem o fluxo do programa, escolhendo caminhos diferentes de acordo com o pretendido.
Fortemente baseadas nas estruturas de outras linguagens têm a seguinte sintaxe:

```
if( condition ){ ... }
elseif( condition ) { ... }
else { ... }
```

O programador pode usar apenas um ***if*** por uma instrução condicional, e adicionar quantos ***elseif*** desejar. No entanto apenas pode utilizar um ***else*** por instrução condicional.

Não pode surgir um ***elseif*** nem um ***else*** sem um ***if*** primeiro. Também não pode surgir um ***else*** seguido de um ***elseif***. Sempre que surge um ***if*** no decorrer do programa presume-se como entrada numa nova instrução condicional.
Condition representa uma ou várias comparações, que são separadas pelas palavras reservadas ***and*** e ***or*** que são respetivamente conjunção e disjunção lógica.
Naturalmente, que sempre que se entra numa instrução condicional, entramos num novo contexto, logo todas as variáveis definidas dentro do mesmo, não serão acessíveis fora dele.

```
#Exemplo de uma instrução condicional
int i -> 24;
if( i<2 or i>25 ){
    i -> 4;      #i é acessível aqui e em todos os contextos
    int c -> 2; #c apenas é acessível aqui
}
elseif(i >= 5 and i <= 45) {
    #Novo Contexto, c não é acessível aqui
    i++;    #incremento de i
}
elseif(stringVar = "Hello" or stringVar!="World"){ #Novo Contexto
    i--;    #Decremento de i
}
else { #Novo contexto
   boolen var -> true;
}
```

3.6 Ciclos

3.6.1 While e Do While

Tal como as estruturas condicionais, ambos estes ciclos dependem de uma condição.

```
while ( condition ) {...} #Sintaxe ciclo while
do { ... } while( condition )  #Sintaxe ciclo do while
```

No caso do ciclo ***while*** a condição é definida no início (determinar ou não a entrada no ciclo) e é verificada sempre que chegamos ao final do ciclo (verificar a continuidade no ciclo).
No ciclo ***do while*** a condição apenas é verificada no final.
Em ambos os casos, se entrarmos dentro do ciclo, significa que estamos a entrar num novo contexto logo todas as variáveis que forem definidas dentro dos mesmos, não serão acessíveis fora deles.

```
#Exemplo de um ciclo while
boolean check -> true;
int i -> 0;
while ( check = true ){ #Caso se verifique a condição continuamos no ciclo
    if(i > 10){
```

```
        check = false;
    }
    else{
        i++;
    }
}
#Exemplo de utilização de um ciclo do while
int i -> 0;
do {
    i++;
} while( i < 10 )#Enquanto se verificar a condição continuamos dentro do ciclo
```

3.6.2 Ciclo For

O ciclo ***for*** apresenta duas sintaxes diferentes.

```
for int var in [limInf, limSup] {...} #Sintaxe do ciclo For
for string s in arraystring {...}  #Sintaxe alternativa  do ciclo for
```

Na primeira sintaxe, define-se uma variável do tipo inteiro e percorre-se um intervalo. Se a variável não for alterada dentro do ciclo, presume-se que ela percorre cada valor inteiro definido no intervalo. Os valores dentro do intervalo, têm de ser do tipo inteiro (podem ser variáveis).
Os parêntesis retos usados para definir o intervalo tanto podem estar abertos como fechados, dependendo do pretendido.

Tabela 3.3: Especificação de intervalos no Ciclo for

Configuração dos Parêntesis	Equivalente em Java
int i in [0, 5]	(int i=0 ; i<=5 ; i++)
int i in [0, 5[	(int i=0; i<5 ; i++)
int i in]0, 5]	(int i=1; i<=5; i++)
int i in]0, 5[	(int i=1; i<5; i++)

A segunda sintaxe serve para quando se quiser iterar sobre um array. O utilizador deve definir uma variável do mesmo tipo que o do array sob o qual pretende iterar. Posteriormente se o array já estiver definido, deve simplesmente indicar o seu nome. Não é aceitado um array não definido ou uma variável que não seja do mesmo tipo que o do array.

```
#Exemplo da primeira sintaxe do ciclo for
for int i in [0, 10] {
    #Variável s não está definida fora do ciclo
    string s -> "i vai de 1 em 1 de 0 até 10";
    s -> "ciclo repete-se 11 vezes";
}
```

```
#Exemplo da segunda sintaxe do ciclo for
int total -> 0;
for int i in arrayint {
    total -> total + i;
} #Em cada iteração i toma um valor diferente do array
```

3.6.3 Instrução break

A instrução ***break*** é utilizada quando o utilizador quiser que um ciclo acabe mais cedo do que o esperado. Esta instrução apenas pode ser usada dentro de ciclos, uma vez por cada contexto definido dentro do ciclo.
De notar que também não é autorizado qualquer instrução após o break.

```
#Exemplo da utilização do break
for int i in [0, 10] {
    if(i = 8){
        i = 0;
        break;  #primeira e única utilização do break neste contexto.
    }
    elseif(i+20 = 29){
        i = 1;
        break; #primeira e única utilização do break neste contexto
    }
}
```

3.7 display e saveFile

A instrução ***display*** funciona como uma função que recebe três argumentos, sendo o primeiro obrigatório e os outros dois facultativos.
Esta instrução serve para apresentar uma pergunta ao utilizador, através de uma interface gráfica que vai permitir que o utilizador responda à pergunta.

```
display(k, mc, 20);
#display da questão k, modo escolha múltipla, 20 segundos para responder
```

O primeiro argumento do método tem de ser uma variável do tipo ***kuest***, já inicializada.
Caso não seja do tipo ***kuest*** ou não tenha sido inicializada então o programador será notificado do erro.
O segundo argumento é a especificação da maneira como o utilizador quer apresentar a pergunta:

- mc (multiple choice) - escolha múltipla;
- tf (true false) - verdadeiro ou falso;
- txt (text) - textual.

O terceiro argumento tem de ser do tipo inteiro (variável obrigatoriamente inicializada ou número) e indica o tempo que o utilizador tem para responder. Se o argumento for válido irá surgir na interface uma contagem decrescente começada no valor indicado.
Como já referido anteriormente o segundo e terceiro argumento, não são obrigatórios, por isso, se o utilizador não especificar o modo como que apresentar a questão, será apresentada por default em modo escolha múltipla. Caso o programador não especifique o tempo, então não surge nenhuma restrição temporal para a resposta à questão, e aparece apenas uma contagem crescente do tempo que o utilizador está a demorar a responder.
De notar também que esta função devolve um inteiro, caso o utilizador necessite de processar o resultado obtido resultante da questão.

- –1 - caso não respondeu;
- >=0 e <=100 - cotação da resposta se respondeu;

```
display(k); #Pergunta k em modo default(escolha múltipla e sem restrições temporais)
display(k, txt); #Pergunta k em modo textual, sem restrições temporais
display(k, tf, 50); #Pergunta k, modo verdadeiro ou falso, 50 segundos para responder
int i -> display(k); #Guardar a cotação obtida numa variável i
```

A instrução ***saveFile*** tal como o ***display*** funciona como uma função, no entanto esta só recebe um argumento e é facultativo. Esta instrução apenas pode aparecer uma vez ao longo de todo o código e se aparecer significa que vai ser criado um ficheiro com feedback resultante da resposta às perguntas que foram efetuadas pelas sucessivas chamadas anteriores à função ***display***.
Esse feedback inclui: as perguntas que foram feitas, as respostas dadas, o tempo que o utilizador demorou a responder a cada resposta, a cotação total, média por pergunta e o tempo total.
O argumento do ***saveFile*** tem de ser do tipo string e representa o nome do ficheiro no qual o utilizador pretende que seja gravada a informação. Se não for especificado nenhum nome, será gravado por defeito num ficheiro intitulado 'resultado.txt'.

```
display(k); #Pergunta k em modo default
saveFile(); #Guardar as informações resultantes da resposta à pergunta k
#Vão ser guardadas num ficheiro intitulado result.txt

saveFile("teste"); #Outra alternativa para uso do saveFile
#Guardar num ficheiro teste.txt
```

3.8 Choose from memory

A instrução ***choose from memory*** serve essencialmente para que o utilizador vá buscar à base de dados, perguntas de acordo com as suas necessidades. Funciona como uma função que pode levar um total de 4 argumentos, todos facultativos, e devolve um ***array kuest***.

```
choose <int> from memory(arguments); #Sintaxe da instrução choose
#<int> representa o nº de perguntas pretendido
#também é facultativo
```

Tabela 3.4: Argumentos da instrução choose from memory

Argumentos	Finalidade
<int>	especificar o nº de perguntas pretendido
theme = <string>	especificar o tema pretendido
ID = <string>	especificar o ID pretendido
numberOfanswers = <int>	especificar o número de respostas pretendido
dificulty = <int>	especcificar a dificuldade das perguntas

Caso o utilizador não especifique nenhum argumento serão copiadas todas as perguntas presentes na base de dados até ao momento.
Se não for especificado o número de perguntas, serão copiadas todas as perguntas que correspondem aos argumentos indicados até ao momento.
O número de respostas é indicado entre a palavra ***choose*** e a palavra ***from***. enquanto que todos os outros são especificados dentro dos parêntesis (pela ordem que o utilizador desejar).
Os argumentos ***theme*** e ***ID*** aceitam para além de strings e variáveis do tipo string, arrays do tipo string.
Todos os argumentos (sem ser o <int>) permitem o operados lógico ***OR*** para que caso seja necessário adicionar mais do que uma especificação do mesmo tipo.

```
array kuest k;
k -> choose from memory(theme="animais", ID="p01", numberOfanswers=4, dificulty=2);
#escolher todas as perguntas do tema animais com ID p01, 4 respostas e dificuldade 2

array kuest k;
k -> choose 4 from memory(theme="cinema" OR "animais", dificulty=1 OR 2 OR 3);
#escolher 4 perguntas dos temas cinema e animais e com dificuldade menor ou igual que 3

array string temas = {"animais", "cinema"};
array string as = {"p01", "p02"};
array kuest k -> choose from memory(theme = temas, ID = as};
#escolher todas as perguntas com tema pertencente ao array temas e ID pertencente a as

#em todos os casos as perguntas escolhidas são guardadas num array de questões k
```

É possível um utilizador guardar o valor de múltiplas operações choose no mesmo array sem o limpar.

```
array kuest k;
k -> choose from memory(theme = "cinema");
k -> choose from memory(theme = "antlr");
#No array vão aparecer primeiro as questões do tema 'antlr'
# e depois as do tema 'cinema'
```

3.9 kuestover

Esta instrução termina um programa antes do final da sua execução. Pode ser usada em qualquer contexto a qualquer altura do programa, exceto a seguir a uma instrução break.

```
int i-> 2;
kuestover;  #programa termina aqui
i -> 3;
```

Capítulo 4

Manual de utilização

4.1 Utilização

Para compilar um programa de ***Kuest***, é necessário seguir os seguintes passos (todos no diretório Compiler na raiz do repositório):

```
#compilação de gramáticas e de todos os ficheiros da linguagem (caso não tenha sido feito)
antlr4-build
#Caso surjam erros, correr este comando uma segunda vez

#Compilar o programa efetuado
java QuizMakerMain nome_ficheiro.kuest

#Compilar o ficheiro resultante da compilação do programa
javac Output.java

#Correr o programa efetuado
java Output
```

Ficheiros que não tenham uma extensão ***.kuest*** não serão aceites.
Caso o utilizador pretenda adicionar ficheiros ***.q*** para adicionar perguntas à base de dados simplesmente necessita de os adicionar ao diretório Compiler, de preferência num subdiretório.
Todos os ficheiros ***.q*** são compilados automaticamente quando se compila um programa ***.kuest***. Ficheiros que não tenham uma extensão ***.q*** não são compilados.
Importante referir que se surgir algum erro resultante da análise semântica feita ao programa desenvolvido pelo utilizador o processo de compilação não vai ser realizado.

4.2 Exemplos desenvolvidos

4.2.1 Utilização de perguntas da memória

Neste exemplo (Figura 4.1), temos a criação de duas variáveis do tipo int, value e time, a primeira tem como função registar a percentagem resultante da cotação da resposta dada

à pergunta e a segunda variável para controlar o tempo, inicializada com o valor 100 (segundos). Além disso, há a criação de um array kuest que irá guardar, neste caso as perguntas (com o seu id, dificuldade, as suas devidas respostas e cotações) com o tema logica ou desporto registados na memória. Seguidamente, é baralhado aleatoriamente as perguntas guardadas no array kuest.

Em questões mais práticas deste exemplo, o utilizador corre todo o array e decide dependendo do número de respostas de cada pergunta como esta irá ser apresentada, se por escolha múltipla (número de respostas maior ou igual a 4) ou como verdadeira ou falso (menor a 4). Além disso, há a verificação da pontuação recebida em cada resposta (value) e se esta for inferior a 40, a variável que representa o tempo (time) diminui 10 valores, ou seja, na pergunta seguinte a variável deixa de começar em 100, mas sim no valor resultante da subtração e assim sucessivamente até o valor time for igual a 0. Quando isso acontecer, o programa acaba e é guardado o resultado do questionário num ficheiro com o nome "report.txt"(este é o ficheiro onde os dados vão ser guardados caso o utilizador não dê dados em contrário).

```
array kuest arrk -> choose from memory(theme = "matematica");
int NumberOfRights -> 0;
int value -> -1;

for int i in [0, arrk.length[ {
    if(NumberOfRights <= 3){
        value -> display(arrk[i], tf, 45);
    }
    elseif(NumberOfRights >= 4 and NumberOfRights<=8){
        value -> display(arrk[i], mc, 55);
    }
    else{
        value -> display(arrk[i], txt, 60);
    }

    if(value > 30){
        NumberOfRights++;
    }
}

saveFile();
```

Figure 4.1: *gtest2.kuest*

Ainda utilizando perguntas da memória, neste exemplo (Figura 4.2) há a criação da variável NumberKuest, que vai ser inicializada com o valor 0. Após isso, é utilizado um ciclo for com o auxilio da variável do tipo kuest, k, que irá guardar perguntas do tema cinema com dificuldade 1, 2 ou 3 e irá imprimir as perguntas desse tema, em formato escolha múltipla, enquanto a variável NumberKuest é menor ou igual a 3. No segundo ciclo for, o resultado será idêntico ao anterior, com a exceção do tema que neste caso é sobre futebol, a dificuldade é 2, 3 ou 4 e o programa continua a mostrar perguntas deste tema enquanto a variável NumberKuest for inferior ou igual 8 (é de notar que a variável NumberKuest entrou neste ciclo for com o valor 4). No terceiro e último ciclo for, o resultado mais uma vez é idêntico aos anteriores, com a diferença do tema que é matemática com dificuldade 3, 4 e 5 e o programa continua a mostrar perguntas deste tema enquanto a variável NumberKuest for inferior ou igual 10 (a variável NumberKuest entrou neste ciclo for com o valor 9).

```
int NumberKuest-> 0;

for kuest k in choose from memory(theme="cinema", dificulty=1 OR 2 OR 3){

    display(k, mc);
    if(NumberKuest > 3){
        break;
    }
    NumberKuest++;
}

for kuest k in choose from memory(theme="desporto", dificulty=2 OR 3 OR 4){

    display(k, mc);
    if(NumberKuest > 8){
        break;
    }
    NumberKuest++;
}


for kuest k in choose from memory(theme="matematica", dificulty=3 OR 4 OR 5){

    display(k, mc);
    if(NumberKuest > 10){
        break;
    }
    NumberKuest++;
}
```

Figure 4.2: *gtest3.kuest*

4.2.2 Criação de perguntas

Neste exemplo (Figura 4.3), são criadas as variáveis k0 e k1, ambas do tipo kuest, com a função de criar perguntas (para mais informação sobre como representar essas perguntas ver o capitulo 2) de modo a que possam ser utilizadas no questionário que o utilizador deseja criar. Em seguida, o utilizador cria um array de kuest, ak, para poder armazenar as perguntas anteriormente escritas. Após isso, decide optar por a utilização de um for para percorrer ak e decidir se faz o display da pergunta ou não, tudo depende se a pergunta em questão é ou não do tema musica, neste caso a pergunta é esse deste tema. Por fim, o programa acaba quando não houver mais perguntas armazenadas em ak e seguidamente guarda num ficheiro a informação do questionário descrito a cima.

```
kuest k0 -> /*musica-P1-2["Quem é o vocalista da banda Nirvana?"]{
    "Dave Grohl" - 0;
    "Kurt Cobain" - 100;
    "Chris Cornell" - 0;
    "Krist Novoselic" - 0;
}
*/;

kuest k1 -> /*musica-P2-5["Que banda foi banida do SNL(Saturday Night Live)?"]{
    "Nirvana" - 0;
    "Rage Against The Machine" - 100;
    "Foo Fighters" - 0;
    "Led Zeppelin" - 0;
}
*/;

array kuest ak;
ak.add(k0, k1);

for kuest k in ak{
    if(k.get(theme) = "musica"){
        display(k);
    }
}

saveFile();
```

Figure 4.3: *gtest6.kuest*

Ainda utilizando perguntas criadas pelo utilizador e adicionando também perguntas da memória, neste exemplo (Figura 4.4 o utilizador opta por criar perguntas da sua autoria, usando as variáveis k0,k1,k2,k3,k4 do tipo kuest (na figura que se segue dá para ver duas dessas perguntas). Posteriormente, é criado um array ak, do tipo kuest, que guarda as perguntas da memória com o tema desporto e que seguidamente é lhe adicionado as perguntas

anteriormente criadas. Por fim, o utilizador decide baralhar as perguntas (isto é conseguido com a linha ak.shuffle). Seguidamente, é criada uma variável value, do tipo int, onde a sua função será guardar a cotação resultante de cada pergunta e se esse resultado for inferior a 50 o programa termina e guarda as informações do programa no ficheiro "Score.txt"(ficheiro que o utilizador escolheu). Além disso, a apresentação de cada pergunta pode mudar, pois se esta tiver dificuldade 4 ou 5, irá ser representada como uma pergunta de verdadeiro ou falso e com um limite de tempo de 50 segundos, mas se por outro lado a dificuldade for diferente da anteriormente mencionada, a pergunta será representada como escolha múltipla e com um tempo limite de 30 segundos.

```
kuest k3 -> /*desporto-id3-5["Quem é o atual campeão mundial de xadrez?"]{
    "Vladimir Kramnik" - 0;
    "Viswanathan Anand" - 0;
    "Magnus Carlsen" - 100;
}
*/;

kuest k4 -> /*desporto-id4-4["Quantas medalhas de ouro conquistou Michael Phelps em Jogos
Olímpicos?"]{
    "15" - 0;
    "17" - 0;
    "28" - 0;
    "23" - 100;
}
*/;

array kuest ak -> choose from memory(theme = "desporto");
ak.add(k0, k1, k2, k3, k4);
ak.shuffle;

int value -> 0;
for kuest k in ak{
   if(k.get(dificulty) = 4 or k.get(dificulty) = 5){
        value -> display(k, tf, 50);
    }
    else {
        value -> display(k, mc, 30);
    }

    if(value < 50) {
        break;
    }
}

saveFile("Score");
```

Figure 4.4: *p1.kuest*

Capítulo 5

Conclusões

Neste trabalho abordámos o tema dos questionários iterativos. Cumprimos todos os objetivos que nos tínhamos proposto, nomeadamente a criação de uma base de dados de perguntas, perguntas das quais caraterizadas com todos os parâmetros pedidos e ainda a adição de um id; a criação de duas linguagens, mais uma vez com todos os parâmetros pedidos; a criação de uma interface de modo a tornar mais fácil o manuseamento do programa em questão; a criação de um ficheiro para guardar os resultados retirados do programa. Por último, este trabalho foi muito importante para o nosso aprofundamento da matéria lecionada nas aulas teóricas e práticas.

Capítulo 6

Contribuições dos autores

Este trabalho foi desenvolvido por 5 alunos da Universidade de Aveiro, no âmbito da cadeira de Linguagens Formais e Autómatos, do curso Engenharia de Computadores e Telemática.

A percentagem de trabalho realizado por cada aluno foi:

- Alexandre Tomás - 20 %
- Ana Filipe - 20 %
- Gabriel Ribeiro - 20 %
- João Gameiro - 20 %
- Pedro Abreu - 20 %