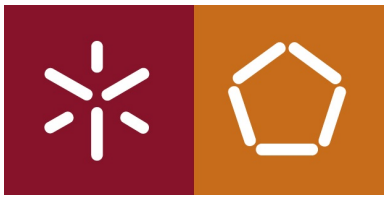

Universidade do Minho

Mestrado Integrado em Engenharia Informática

Laboratório em Engenharia Informática - 4º ano

DSL para a Geração de Chatbots

Diana Ribeiro Barbosa, a78679
Francisco José Moreira Oliveira, a78416
Raul Vilas Boas, a79617
25 de Junho de 2019

Resumo

O presente trabalho foi desenvolvido no âmbito da unidade curricular **Laboratório em Engenharia Informática** do 4º ano do Mestrado Integrado em Engenharia Informática da Universidade do Minho, que tem como objetivo a aprendizagem prática de conceitos de desenvolvimento de um projeto associado a um perfil de especialização, no caso, Processamento de Linguagens e Conhecimento. A supervisão do mesmo ficou a cargo dos professores José João Almeida e Pedro Rangel Henriques.

O projeto desenvolvido consiste de uma linguagem (*Domain Specific Language*) que permite especificar *chatbots* e de um sistema capaz de os gerar automaticamente com base numa dada especificação (com fontes de informação associadas) e também de lidar com toda a coordenação dos mesmos necessária à interação com o utilizador, isto é, um "orquestrador de chatbots"que analisa o input do utilizador e procede à escolha da melhor resposta/ação a ser devolvida/realizada.

A criação destes componentes foi realizada com sucesso tal como a sua aplicação a casos de estudos reais, como é o caso da Semana da Engenharia Informática da Universidade do Minho. Foi ainda possível desenvolver várias funcionalidades interessantes em cada um deles (e.g. criação de estados e tipos de comportamento dos *chatbots*) que tornaram o sistema mais robusto, realista e aplicável numa grande variedade de contextos.

Neste relatório introduzimos o tema, elaborando sobre o seu enquadramento, contexto, problema a nos propomos resolver, objetivos e os resultado obtemos. Damos a conhecer também o processo de conceção da resolução, testes e casos de estudo que demonstram a sua eficácia e fazemos uma apreciação crítica e ideias para um possível trabalho futuro.

Conteúdo

Lista de Figuras

Capítulo 1

Introdução

O projeto desenvolvido denomina-se DSL para a Geração de Chatbots e, tal como o nome indica, a sua finalidade é a criação de um sistema que recebendo uma especificação (seguindo as regras sintáticas, léxicas e semânticas de uma linguagem por nós definida) e dada uma fonte de informação, gere um ou ou mais chatbots capazes de manter um diálogo, prestar serviços informativos, entre outras funcionalidades.

Deste modo, podemos distinguir alguns componentes de relevância cujo desenvolvimento consideramos fundamental à essência da ferramenta. Entre eles, um motor geral de conversação baseado em regras padrão - função respondedora (isto é, a capacidade de receber regras específicas de comportamento definidas pelo utilizador como também pré-determinadas), uma Calculadora de Regras de Chatbots que a partir de uma variedade de recursos (como por exemplo listas de provérbios, bases de dados, ontologias) construa chatbots. Por fim, a capacidade de somar e subtrair os programas criados, ou seja, os chatbots que estarão ativos e poderão interagir com o utilizador.

Assim sendo, temos vários conceitos relevantes associados ao tema cujo entendimento é essencial à compreensão do projeto, nomeadamente, "DSL", "Processamento de Linguagem Natural"e "Chatbot". O primeiro é uma abreviação de Domain Specific Language (Linguagem de Domínio Específico) que, tal como o nome indica, é uma linguagem de especificação particular a um domínio concreto (eg. HTML para páginas web e SQL para bases de dados relacionais). Em oposição, temos o conceito de GPL (General Purpose Languagens) como é o caso de C, Java, Python entre outras linguagens de programação, aplicáveis a vários domínios.

Em segundo lugar, temos o conceito de Processamento de Linguagem Natural. Este é uma sub-área de Ciências de Computação que visa a análise, estudo e interpretação de linguagem natural (humana) com o objetivo de tornar possível a uma máquina a sua deteção, compreensão e geração automática. Este campo atualmente está relacionado com as áreas de Inteligência Artificial e Engenharia de Informação devido às suas possíveis aplicações práticas.

Por fim, é importante clarificar o conceito de Chatbot. Um chatbot é um programa desenvolvido com o objetivo de simular uma conversa com um utilizador humano, respondendo ao mesmo com frases em linguagem natural, seguindo a linha de conversa de forma sintaticamente correta e o melhor possível de acordo com o que lhe foi dito.

A capacidade de identificar a intenção do utilizador e de extrair a informação pertinente do input são dos fatores mais relevantes a uma boa performance do programa. Após esta primeira análise, o chatbot providenciará um output que pode ser a resposta à pergunta que lhe foi feita (caso se trate de uma), uma resposta informativa em relação ao assunto, uma pergunta que o ajudará a tentar perceber melhor o que lhe foi dito ou uma resposta pré-definida caso as outras opções não sejam possíveis. No fundo, estes podem ser criados de forma a serem produtivos, isto é, serem capazes de providenciar informação com rapidez, exatidão e eficácia mas também numa vertente de entretenimento.

1.1 Chatbots: Passado, Presente e Futuro

1.1.1 Origem e primeiros chatbots

A história dos chatbots começa em 1950 com um artigo (*Computing Machinery and Intelligence*) da autoria de Alan Turing no qual ele teoriza que, para uma máquina poder ser considerada verdadeiramente inteligente, esta deve ser indistinguível de um ser humano numa conversa textual, isto é, a pessoa com quem o programa está a tentar conversar - em tempo real - não ser capaz de distinguir se está a falar com uma máquina ou com um indivíduo. Este critério de inteligência ficou conhecido como teste de Turing e tornou-se um marco e fonte de inspiração e discussão no estudo e desenvolvimento de chatbots e Inteligência Artificial na comunidade académica até aos dias de hoje.

Vários anos depois (1966) surge um dos primeiros chatbots, o *ELIZA*. Criado por Joseph Weizenbaum no MIT, o *ELIZA* fazia-se passar por um psico-terapeuta especializado em terapia não diretiva / abordagem centrada na pessoa. [1] Esta foi a abordagem escolhida para o bot uma vez que numa entrevista psiquiátrica perguntas e respostas que noutras situações seriam consideradas ilógicas e pouco razoáveis, são mais aceitáveis e até mesmo esperadas. Assim, as ideias pré-concebidas e expectativas do utilizador acerca do discurso de um psico-terapeuta, ajuda a tornar o bot mais credível. Um exemplo que demonstra adequadamente esta teoria é o seguinte: imaginemos que um paciente durante uma consulta diz "Foi um bom passeio de barco", ao que o terapeuta responde "Fala-me de barcos". O cenário mais provável é o paciente assumir que a mudança de direção efetuada tem um propósito e não que o terapeuta não sabe nada acerca do assunto.

A nível do seu funcionamento, o *ELIZA* procura reconhecer *keywords* e padrões *pattern-matching based* e responde com frases pré-programadas e através de métodos de substituição, tentando prolongando a conversa e dando a ilusão de compreender o utilizador. Apesar de ter sido um caso de sucesso para a época e ter conseguido enganar os utilizadores numa elevada percentagem de casos, falhou o teste de Turing. No entanto, o seu método de construção serviu (e ainda serve) de inspiração para muitos programadores.

[1] *"Person-centered therapy uses a non-authoritative approach that allows clients to take more of a lead in discussions so that, in the process, they will discover their own solutions. The therapist acts as a compassionate facilitator, listening without judgment and acknowledging the client's experience without moving the conversation in another direction. The therapist is there to encourage and support the client and to guide the therapeutic process without interrupting or interfering with the client's process of self-discovery."* [3]

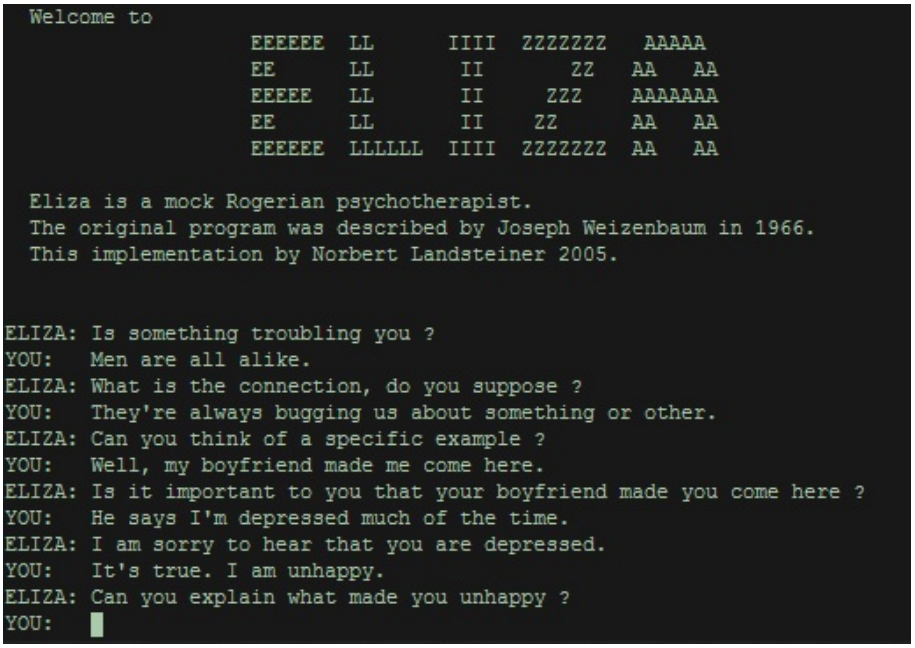

Figura 1.1: Exemplo de Funcionamento do *ELIZA*

Outros *chatbots* que ficaram conhecidos ao longo dos anos, podemos destacar o *Parry* (1972, tentativa de melhorar o ELIZA), *Jabberwacky* (1988, simular conversas humanas humorísticas e interessantes para entretenimento), *Dr Sbaitso* (1992), *ALICE* (1995, *Artificial Linguistic Internet Computer Entity*), *Smarterchild* (2001, considerado precursor da Siri e do SAMSUNGS Voice), IBM's WATSON (2006, inicialmente criado com o objetivo de competir no programa de televisão Jeopardy, conseguiu vencer a dois campeões do jogo).

1.1.2 Chatbots na atualidade

Os *chatbots* mais conhecidos nos dias de hoje são a Siri e a Alexa, ambos capazes de reconhecer comandos auditivos e enquadrados na categoria de assistentes pessoais. Em Portugal, podemos destacar o Sofia, assistente virtual da TAP.

No entanto, há imensos *chatbots* na Web, inovadores úteis e com bastante sucesso. A título de exemplo, o bot *U-REPORT* tem tido um impacto significativo em muitas comunidades permitindo aos seus residentes transmitir as necessidades e problemas mais urgentes/preocupantes que os afetam (e.g. dados recolhidos na Libéria através do *U-REPORT* levaram à descoberta de que os professores de 83% das 13000 crianças sondadas exigiam favores sexuais em troca de boas notas, o que levou a que a UNICEF em conjunto com o Ministério da Educação da Libéria tomassem medidas para travar a situação como consequência da informação recolhida pelo mesmo).

Numa outra área mas também com grande potencial para um impacto positivo, é possível destacar o MedWhat. Este bot utiliza elevados volumes de pesquisas médicas e relatórios científicos da área para efeitos de diagnóstico médico, tornado-os mais rápidos, transparentes tanto para pacientes como para os médicos.

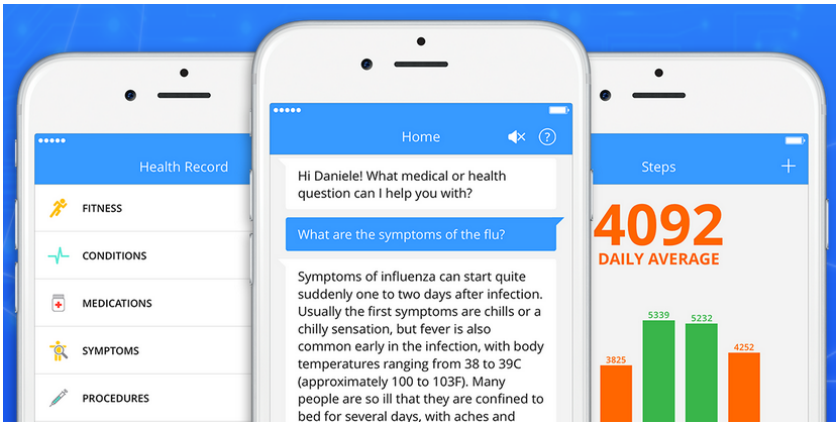

Figura 1.2: Exemplo de Funcionamento do *MedWhat*

Por outro lado, nem todos os chatbots têm como objetivo serem informativos e de utilidade, sendo vários os casos de chatbots para efeitos de entretenimento ou apenas para estudo de áreas como Inteligência Artificial. Este é o caso de um chatbot da Microsoft de nome Tay. Este foi criado para funcionar no Twitter, aprendendo com as publicações da plataforma de modo a tornar o seu discurso mais human-like. No entanto, tal com vários outros casos do género, este teve que ser removido da web uma vez que se tornou racista e homofóbico.

De facto, este tipo de programas podem ser aplicados a uma grande variedade de áreas e têm crescido bastante no mercado devido às diversas vantagens que trazem às empresas. Entre elas:

- redução de custos a nível de recursos humanos já que um sistema destes tem a capacidade de executar várias tarefas que de outro modo teriam que ser feitas por pessoas;
- disponível 24 horas por dia, 7 dias por semana;
- atender vários clientes ao mesmo tempo eliminando o tempo de espera;
- permitem prestação de serviços em diversas plataformas ao mesmo tempo, como por exemplo, nas redes sociais;
- personalização da personalidade do mesmo (método de reação), sendo completamente controlado;
- capacidade de realizar um elevado volume de tarefas num curto espaço de tempo o que permite agilizar o funcionamento de uma empresa.

1.1.3 Futuro dos chatbots

A popularidade dos *chatbots* tem aumentado com o tempo sendo cada vez maior o número de empresas que os pretende incluir no seu modelo de negócios. A CEO da Microsoft, Satya Nadella, prevê que nos próximos cinco a dez anos, todas as marcas possuam um assistente digital para as representar. [2]

De facto, prevê-se o aumento do volume de serviços realizados via *chatbots* através de conversas, em particular em plataformas de redes sociais, já que estas são utilizadas por uma percentagem bastante considerável dos consumidores. [1]

1.2 Motivação e objetivos

O objetivo deste projeto prende-se com a criação de um sistema de geração automática de chatbots dada uma especificação e uma ou mais fontes de informação, sob a forma de uma Domain Specific Language criada para o efeito. De facto, a motivação por detrás do mesmo passa por ser sistema inovador uma vez que não existem ainda linguagens que o consigam fazer. A nível de projetos semelhantes, podem-se referir duas linguagens de programação de *chatbots*, nomeadamente, AIML e SIML.

AIML (*Artificial Intelligence Modelling Language*) é uma linguagem de markup baseada em XML que tem como objetivo a criação de aplicações de inteligência artificial. Deste modo, possui várias tags disponíveis, dentro das quais se definem regras de comportamento de *chatbots*. Esta é uma linguagem de fácil compreensão e simples de programar. No entanto, não possui a capacidade de gerar automaticamente *chatbots* bastando apenas indicar um dos diversos tipos de comportamento definidos, um valor de prioridade e um ficheiro com informação num dos vários formatos permitidos, tal como o nosso sistema se propôs e efetivamente consegue fazer.

Por outro lado, SIML (Synthetic Intelligence Markup Language) é considerada uma linguagem mais poderosa com várias ferramentas disponíveis, entre elas um interpretador (Syn Bot - SIML Interpreter), conversor de AIML para SIML, análise de código para otimização (sinalizando repetições e código desnecessário, ...) entre outras. Consegue ainda criar *chatbots* mais realistas do que a sua competidora. No entanto, possui uma maneira de operar e objetivo diferente da DSL por nós criada, da mesma forma que AIML.

Assim, o projeto por nós desenvolvido é inovador na sua área e destaca-se pela facilidade de criar automaticamente *chatbots* sem necessidade de grandes conhecimentos de programação. De facto, basta apenas garantir que as fontes de informação se encontram com o formato e estrutura adequada a cada tipo de bot que se pretende gerar, e que a especificação se encontra corretamente escrita (para este objetivo o sistema recorre a uma gramática que avisa quanto a e explicita possíveis erros), o que se revela fácil uma vez que a linguagem é fácil de compreender e utilizar.

1.3 Estrutura do relatório

Neste relatório, após a introdução ao tema sobre o qual nos debruçaremos (o seu enquadramento, contexto, motivação e objetivos), elaboramos no capítulo 2 sobre a análise e especificação do problema, nomeadamente, fazemos uma descrição informal deste seguida pela análise do mesmo.

No capítulo 3, revelamos o processo de conceção e desenho da resolução obtida, concretamente, explicamos a linguagem que foi desenvolvida a fundo, nomeadamente, os tipos de *chatbots* pré-definidos bem como os diversos estados de comportamento e a gramática criada para a verificação das especificações futuras.

No capítulo 4 debruçamo-nos sobre o funcionamento do sistema, quer a nível da geração dos *chatbots* a partir da DSL, como também do processo de escolha da resposta a devolver e algumas funcionalidades extra que conseguimos implementar com sucesso.

O capítulo 5 é dedicado aos testes do sistema. Nele explicamos como é possível realizar a instalação e compilação do mesmo, bem como a estrutura do sistema. Expomos ainda decisões tomadas ao longo do processo de desenvolvimento, possíveis alternativas à atual implementação e problemas que decorreram ao longo da mesma. Por fim, mostramos alguns testes feitos e os resultamos, bem como um caso de estudo ao qual aplicamos o trabalho, nomeadamente, a Semana da Engenharia Informática Universidade do Minho.

Por último, no capítulo 6 (Conclusão) fazemos uma síntese do trabalho, expomos o estado final do projeto, fazemos também uma análise crítica dos resultados e, por fim, indicamos algumas ideias e sugestões quanto a um possível trabalho futuro.

Capítulo 2

Análise e Especificação

O projeto sobre o qual nos debruçamos ao longo deste relatório teve como base um problema. Este será especificado e analisado no presente capítulo, concretamente, procederemos a uma descrição informal do mesmo, seguida de uma análise mais aprofundada.

2.1 Descrição informal do problema

O objetivo do projeto detalhado neste relatório consiste da criação de um sistema que permita a geração automática de *chatbots* dada uma especificação (seguindo uma linguagem desenvolvida para o efeito) e uma ou mais fontes de informação. Este sistema necessita de ser capaz de gerar um ou mais chatbots com capacidade de manter um diálogo, prestar serviços informativos, entreter o utilizador, realizar pequenas tarefas, entre outras funcionalidades. Em suma, deve receber uma ou mais frases de um utilizador humano e responder da forma mais pertinente e adequada possível tendo em conta a especificação que lhe foi dada, quer a nível da relevância da resposta de cada um dos bots (calculada através da prioridade destes definida na especificação e também do contexto e perguntas que se espera que sejam capazes de responder), como também da informação que têm disponível e do tipo de comportamento que se espera deles (passível de ser definido através da linguagem).

2.2 Análise do Problema

Tendo por base a descrição do problema previamente detalhada, seguimos para a sua análise. O problema em questão pressupõe a criação de vários componentes essenciais ao cumprimentos das funcionalidades a que se propõe oferecer, nomeadamente, a criação de uma DSL (*Domain Specific Language*), isto é, uma linguagem específica ao domínio sobre o qual nos iremos focar (i.e. geração automática de chatbots) que deve estar preparada para receber a especificação das fontes de informação para cada um dos bots, para além de outros detalhes relativos ao seu funcionamento. Necessita ainda de um componente que realize a leitura da DSL e produza os programas (chatbots) que irão gerar respostas de acordo com a sua especificação e informação que têm disponível. Por fim, é necessário um componente que escolha uma resposta para devolver ao utilizador de entre as várias sugeridas pelos bots criados.

Resumindo, o problema engloba todo o processo desde a criação dos mecanismos de resposta, definição do comportamento esperado do sistema, exploração do conhecimento e informação disponibilizado, à análise do input do utilizador e escolha da melhor fala do sistema para essa entrada.

Capítulo 3

Concepção/desenho da Resolução

Após analisar o problema, vamos precisar de ter uma linguagem específica que permita descrever os vários *chatbots* e a maneira como estes se interligam. O sistema completo será desenvolvido a partir dessa DSL. Neste capítulo iremos debruçar-nos sobre a mesma, concretamente, as decisões tomadas à cabeça relativamente à solução a desenvolver.

3.1 DSL a desenvolver

Tal como mencionado previamente ao longo deste relatório e de acordo com o próprio nome do projeto (*"DSL para a geração de* chatbots *"*) foi desenvolvida uma *Domain Specific Language* que permite a especificação dos *chatbots* a serem gerados. Concretamente, esta permite detalhar os seguintes parámetros:

- **Tipo do *chatbot* a ser criado:** Existem sete tipos diferentes de chatbots programados, prontos a serem utilizados, sendo estes: `bot_csv, bot_exp, bot_FAQ, bot_lista, bot_QA, bot_tradutor` e `bot_wiki`. Cada um destes possui uma forma distinta de extração de informação e construção de resposta, bem como finalidades únicas. Variam ainda no tipo de ficheiros de dados que aceitam bem como requisitos extra (e.g. `bot_csv` necessita que lhe seja passado um *schema* da informação que recebe). Definido no formato *"CREATE"+tipo_chatbot* (e.g. `CREATE bot_tradutor`). [Informações mais detalhadas em 3.1.1.]

- **Fonte de informação que lhe está associada:** O sistema aceita três formatos distintos de ficheiros que contêm a informação necessária para os bots, nomeadamente, `csv, txt` e `json` (formatos aceites estão associados e variam conforme o tipo do bot). Definido no formato *"FROM"+ficheiro_informação* (e.g. `FROM lusiadas.txt`).

- ***Schema* da fonte de informação (quando aplicável):** *Chatbots* do tipo `bot_csv` requerem um *schema* dos dados que lhe são fornecidos. Definido no formato *"WITH"+ficheiro_schema* (e.g. `WITH agenda_SEI_schema.json`).

- **Nível de prioridade associado a cada *chatbot*:** Valor entre 0 a 5 que ajuda ao cálculo da escolha da resposta a ser devolvida ao utilizador. Quanto maior for o valor deste campo, maior a probabilidade do conteúdo produzido por este bot ser escolhido em prole de outro. Definido no formato *"!"+número* (e.g. `!3`).

- **Estados de comportamento passíveis de serem ativados:** Neste momento encontram-se definidos dois tipos de estados extra, isto é, regras de comportamento que regem o comportamento geral do

sistema a nível da interação com o utilizador. Estes são "chateado"e "informativo". Um dado estado é ativado através de uma ou mais frases por parte do utilizador, por exemplo, este proferir insultos leva à ativação do estado chateado.

3.1.1 Tipos de *chatbots*

Existem sete tipos diferentes de *chatbots*, cada um com funcionamentos e requisitos diferentes. São os que se seguem.

- `bot_csv:` Este tipo de *chatbot* possui um caráter informativo. Quando ao seu funcionamento, aceita ficheiros do tipo csv e necessita de um ficheiro json com o schema (nome da coluna, tipo - *Objeto, Local, Temporal, Pessoal* - e sinónimos) da informação que lhe é fornecida. O seu processo de obtenção de respostas consiste da deteção do tipo de pergunta e do reconhecimento de entradas do csv na pergunta do utilizador. Elaborando, o sistema procura na frase por um elemento do csv e identifica a que linha e coluna pertence. Como resposta irá devolver os elementos cujo tipo seja o mesmo que foi detetado na pergunta, isto é, caso a pergunta seja do tipo "Quando", serão devolvidos os elementos do tipo temporal (e.g. datas, horas, etc).

- `bot_exp:` Este tipo de *chatbot* aceita ficheiros tipo json nos quais cada entrada contem uma lista de expressões e uma lista de respostas associadas. Deste modo, quando o utilizador disser algumas das expressões o sistema responderá com a resposta adequada. Temos como exemplo o bot das saudações e das despedidas.

- `bot_FAQ:` Este tipo de *chatbot* foi feito para receber FAQs dos mais variados domínios e funciona de forma semelhante a um `bot_exp`. Deste modo, recebe um ficheiro json no qual cada entrada é uma lista de uma ou mais perguntas seguida da resposta adequada às mesmas.

- `bot_lista:` Este tipo de *chatbot* recebe ficheiros do tipo txt e procura padrões de semelhança entre o input do utilizador e as frases que possui (o seu nome deriva portanto do facto de ter como fonte de informação associada um conjunto de frases, sejam estas adivinhas, provérbios, falas de um livro, etc).

- `bot_QA:` Este tipo de *chatbot* recebe ficheiros do tipo txt que contêm quádruplos com o tipo da questão, o verbo, as keywords e a resposta (e.g. "`Quem;foi;[primeiro,rei,Portugal];D.Afonso Henriques`"). O processo de procura de respostas consiste de procurar as linhas cujo tipo de pergunta ("quem", "quando", ...) esta presente na pergunta, bem como o verbo (e.g. "foi") seja o mesmo. Nas entradas que cumprirem este requisito, é calculado um rácio que é dado pela divisão do número de keywords em comum entre a pergunta do utilizador e a do ficheiro, a dividir pelo número de keywords da dada pergunta do ficheiro. A resposta a ser devolvida é aquela que o rácio seja maior, isto é, a resposta para a qual houver o maior match entre o que o utilizador perguntou e as keywords daquela entrada no ficheiro. Para o caso de haver mais do que uma pergunta com o mesmo valor calculado de rácio (recordemos que este valor já só é calculado para as perguntas com o mesmo tipo e verbo que a do utilizador), passa-se ao processo de desempate. Este passa por verificar a raridade das *keywords* (recorrendo a um dicionário `dicRank` que contém o número de vezes que uma dada palavra aparece num corpus [1] sobre o qual a contagem foi feita). Quanto mais baixo o valor mais rara é a palavra. Dessas possíveis respostas (todas com o mesmo valor de rácio) a que contiver mais palavras raras é considerada como a que tem maior probabilidade de estar correta, e escolhida sobre as outras.

[1] Corpus linguístico é um conjunto de textos escritos e registos orais numa determinada língua e que serve como base de análise sobre a mesma.

- `bot_tradutor:` Este tipo de *chatbot* recebe ficheiros do tipo json com traduções de palavras para várias linguas. Deste modo, quando ocorrerem frases do género *"Como se diz ... em ...?"*, *"Qual é a tradução de ... em ...?"* entre outras, ele procura pela tradução da palavra no dicionário json que recebeu ou através do módulo que importou.
- `bot_wiki:` Este tipo de *chatbot* recebe ficheiros do tipo json que contêm para cada palavra, a primeira linha da wikipedia com a sua explicação. Este bot é reage a frases do tipo *"O que sabes sobre ...?"*, ou ainda *"Fala-me sobre ...?"*, entre outras e devolve a informação que tem sobre o conceito em questão.

3.1.2 Estados de comportamento

Estados de comportamento no contexto deste projeto estão relacionados, tal como o nome indica, com o comportamento dos *chatbots*. De facto, o uso de certas expressões por parte do utilizador levam o bot a entrar ou sair de um dado estado, aplicando ou retirando consequentes restrições ao seu comportamento.

Estado *normal*

Este é o estado de comportamento *default* do sistema. A partir deste estado pode passar para um dos outros através de *triggers* sob a forma de certas expressões ocorrem nas frases proferidas pelo utilizador. O estado normal permite o acesso a todas as funcionalidades exceto as funcionalidades correspondentes do estado *chateado*.

Estado *chateado*

Este estado ocorre quando o utilizador insulta o *chatbot*. Quando isto acontece, o *chatbot* responderá sempre *"Não falo mais contigo,"* até receber um pedido de desculpas do utilizador. O reconhecimento das ofensas é feito recorrendo a expressões regulares (e.g. *"És mesmo"* seguido de um insulto da lista a que o sistema tem acesso).

Estado *informativo*

O sistema entra neste quando o utilizador diz frases do género de *"Diz-me uma coisa."*, *"Preciso de informações sobre..."* entre outras. Os tipos de *chatbot* que podem responder quando este estado está ativo são o `bot_wiki`, `bot_tradutor`, `bot_csv`, `bot_QA` e o `bot_FAQ`. O estado reverte de informativo para normal quando o utilizador indica verbalmente que já obteve as informações de que necessitava.

3.1.3 Gramática

De modo a garantir a correta utilização da linguagem por parte do utilizador, foi desenvolvida uma gramática para a verificação sintática de todas as entradas. Para tal, recorreu-se à ferramenta ANTLR (*ANother Tool for Language Recognition*) na versão 4.7.1 que trata do parsing do ficheiro `dsl.txt` onde é feita a especificação. Caso esta não esteja de acordo com as regras definidas, o sistema devolve a mensagem `"DSL está mal estruturada. Por Favor corrija-a para continuar."` juntamente com o log de erros. Caso contrário, isto é, a especificação não contenha erros, será devolvida a frase `"DSL compilada corretamente."`.

A gramática criada é a que se segue:

```
grammar gramatica;

dsl: create_block NEWLINE+ states NEWLINE* EOF;

create_block: (create_bot NEWLINE)+ ;
create_bot: PRIORIDADE 'CREATE' BOT_TYPE ('WITH' dataset)? 'FROM' dataset ;

states: 'STATES' STATE+ ;

dataset: STRING '.' FILE_TYPE ;

////////////////////////////////////////////////////////////////////////////

BOT_TYPE: 'bot_csv' | 'bot_lista' | 'bot_wiki' | 'bot_QA'
        | 'bot_exp' | 'bot_FAQ' | 'bot_tradutor';
FILE_TYPE: 'csv' | 'txt' | 'json' ;
STATE: 'CHATEADO' | 'INFORMATIVO' ;

PRIORIDADE: '!'[0-5] ;

STRING: [a-zA-Z0-9_\-]+ ;

NEWLINE:  '\r'? '\n' ;
WS:       [ \t]+ -> skip ;
```

Recordemos que a especificação dos chatbots é definida no ficheiro `dsl.txt` na pasta `dsl`. De acordo com a gramática definida e assumindo que as fontes de informação são as adequadas, uma possibilidade deste ficheiro seria o que se segue:

```
!4 CREATE bot_csv WITH agenda_SEI_schema.json FROM agenda_SEI.csv
!1 CREATE bot_lista FROM lusiadas.txt
!1 CREATE bot_lista FROM proverbios.txt
!5 CREATE bot_wiki FROM wiki.json
!5 CREATE bot_QA FROM portugal_QA.txt
!5 CREATE bot_exp FROM saudacoes.json
!5 CREATE bot_exp FROM despedidas.json
!4 CREATE bot_FAQ FROM FAQ_SEI.json
!3 CREATE bot_exp FROM respostas.json
!3 CREATE bot_tradutor FROM dicionario.json

STATES CHATEADO INFORMATIVO
```

Capítulo 4

Desenvolvimento do sistema

De modo a compreender o funcionamento do sistema, é necessária a compreensão de dois processos pilares do mesmo, concretamente, o processo de geração dos *chatbots* a partir da especificação dos mesmos recorrendo à DSL desenvolvida e, não menos importante, o processo de escolha da resposta a devolver ao utilizador em cada uma das interações.

Neste capítulo explicaremos ambos os processos, bem como algumas funcionalidades extra (não cruciais à solução do problema, mas que consideramos uma mais valia) que foram desenvolvidas.

4.1 Geração dos *chatbots* a partir da DSL

A geração dos *chatbots* a partir da especificação indicada no ficheiro `dsl.txt` (caso o utilizador não pretenda a opção *default* pode usar a flag `--dsl` e indicar uma alternativa) é maioritariamente da responsabilidade de de duas funções, `read_dsl` e `change_tuplos`.

A primeira - `read_dsl` - recebe o ficheiro que contém a especificação (mencionado do paragrafo anterior) e devolve os estados lá indicados (e.g. informativo) e um conjunto de tuplos. Estes contêm o tipo do bot a ser criado (e.g. bot_wiki), as fontes de informação que lhes estão associadas e a sua prioridade.

Consideremos o seguinte exemplo. Para a linha abaixo o tuplo criado seria o seguinte:

```
!4 CREATE bot_csv WITH agenda_SEI_schema.json FROM agenda_SEI.csv
```

Campos do tuplo:

1. tipo de bot: `bot_csv`
2. datasets: `agenda_SEI_schema.json e agenda_SEI.csv`
3. prioridade do bot: `4`

Por fim, a função `change_tuplos` recebe os tuplos criados e adiciona-lhes o conjunto de regras que definem cada tipo de *chatbot*. Esta adição é feita através da função `get_regras` e estas encontram-se definidas em `get_regras.py`. Um exemplo de regras associadas a um tipo de bot é o que se segue.

```
regras_bot_csv = [
    (['NORMAL','INFORMATIVO'],3,r'(.*)',
            lambda x,dataset: bot_csv.responde(x.group(0),dataset[0],dataset[1])),
]
```

Atentemos ao código das duas funções mencionadas, `read_dsl` e `change_tuplos` para uma melhor compreensão do processo descrito.

```
# lê a especificação e retorna uma lista de tuplos que contêm os bots e o dataset a ser usado
def read_dsl(ficheiro):
    triplos = []
    estados = []

    content = open(ficheiro).read().split('\n')
    for linha in content:
        linha = linha.split(' ')
        if 'STATES'==linha[0]:
            estados = linha[1:]
        elif re.match(r'!([0-5])',linha[0]):
            prioridade_bot = int(linha[0][1])
            datasets = []
            for i in range(1,len(linha)-1):
                if linha[i] == 'CREATE':
                    bot = linha[i+1]
                if linha[i] == 'WITH':
                    datasets.append(linha[i+1])
                if linha[i] == 'FROM':
                    datasets.append(linha[i+1])
        if datasets != [] and bot != []:
            triplo = tuple((bot,datasets,prioridade_bot))

            triplos.append(triplo)

    return triplos,estados

#adiciona ao tuplos (tipo_do_bot,dataset,prioridade) as regras associadas a esse tipo de bot
def change_tuplos(triplos_dsl):
    tuplos = []
    for bot,dataset,prioridade_bot in triplos_dsl:
        regras = get_regras(bot)
        tuplo = tuple((regras,dataset,bot,prioridade_bot))
        tuplos.append(tuplo)
    return tuplos
```

4.2 Processo de escolha da resposta a devolver

Para todas as interações do utilizador com o sistema, há um processo de escolha de resposta que ocorre com o intuito de garantir a resposta mais adequada.

Primeiramente, identifica-se o estado ativo do sistema (e.g. normal, chateado,...) e verifica-se se o input do utilizador levou à sua alteração. Caso isto tenha ocorrido, procura-se saber qual é resposta mais adequada de entre as listadas (em respostas_alteracao_estados). Por exemplo, caso o estado tenha sido alterado de chateado para normal, verifica-se se o que o utilizador disse se encontra nessa lista numa das linhas relacionadas com essa mudança de estados. Caso isso seja verdade, devolve-se a resposta lá indicada. Por exemplo, caso o estado tenha mudado de chateado para normal, e o input do utilizador tenha sido *"Desculpa"*, a resposta do bot será *"Estás desculpado."*, de acordo com a seguinte entrada da lista: `(('CHATEADO','NORMAL',r'Desculpa','Estás desculpado.'))`.

Caso não tenha ocorrido uma alteração do estado ativo, vai ser criado uma lista de respostas possíveis tendo em conta os bots que podem operar no estado em questão. É então produzida uma lista de respostas

do tipo (resposta, confiança, bot) que contêm todas as respostas sugeridas pelos *chatbots*. É feita uma ordenação das respostas possíveis e devolvida aquela cuja confiança é maior. O cálculo da confiança é dado pela seguinte expressão: *confianca = (prioridade_bot + prioridade_regra) * ratio*. O valor de *ratio* é devolvido por cada um dos bots e diz respeito à proporção com que se consegue relacionar uma frase dita pelo utilizador com umas das perguntas/entradas de informação a que cada bot tem acesso. O valor de prioridade do bot é o definido na especificação inicial (na DSL) e, por fim, a prioridade da regra em que a interação se encaixa é definida em `get_regras.pt` (na lista de cada uma das regras de cada tipo de bot).

Em ambos os casos (mudança ou não de estado), caso não seja encontrada uma resposta considerada adequada, é sorteada uma da coleção *clueless*. Alguns exemplos destas frases são: *"Não estou a perceber nada...", "O quê?", "Tens a certeza que sabes falar português?!", "Andaste a beber?!", "Repete lá isso de forma que eu entenda."*.

A implementação da função responsável pelas respostas é a que se segue.

```
def responde(input_utilizador,tuplos,estados):
  lista_respostas = []

  global state_atual

  state_anterior = state_atual
  state_atual = altera_estados(input_utilizador,estados,state_atual)

  if state_anterior != state_atual:
    for state_antes,state_depois,regra,resposta in respostas_alteracao_estados:
      if state_antes == state_anterior and state_depois == state_atual:
        match = re.match(regra,input_utilizador,re.IGNORECASE)
        if match is not None:
          return resposta
    return random.choice(clueless)
  else:
    for regras,dataset,bot,prioridade_bot in tuplos:
      for lista_estados_validos,prioridade_regra,regra,funcao, in regras:
        if state_atual in lista_estados_validos:
        # só acontece se o estado atual for permitido na regra
          match = re.match(regra,input_utilizador)
          if match is not None:
            if callable(funcao):
              try:
                resposta,ratio = funcao(match,dataset)
                if resposta is not None:
                  if ratio > 1: ratio = 1
                  confianca = trunc((prioridade_bot + prioridade_regra) * ratio)
                  tuplo = tuple((resposta,confianca,bot))
                  lista_respostas.append(tuplo)
              except:
                pass
    if lista_respostas:
      return lista_respostas[0][0]
    else:
      for estado_check,funcao_estado in regras_estado_resposta:
        if state_atual == estado_check:
          resposta = funcao_estado()
          return resposta

    # sugerir uma adivinha
    return random.choice(clueless)
```

4.3 Funcionalidades extra

4.3.1 Logs - registo das conversas

Cada conversa realizada entre o utilizador e o sistema é registada no ficheiro `log.txt` na pasta `log`. Esta funcionalidade foi pensada com vista a permitir a análise das interações de modo a potenciar a identificação de elementos a serem melhorados. Para cada entrada é identificado a quem pertence a fala e cada conversa encontra-se separada da seguinte. Segue-se um exemplo de uma entrada no ficheiro de logs.

```
User: Olá
Bot: Saudações! Como vai isso?
User: sim e contigo?
Bot: Mais vale o sim tardio, que o não vazio
User: As galinhas tem dentes?
Bot: Quem faz de si, lixo, pisam-no as galinhas
User: Adeus
Bot: Adeus companheiro!

---FIM DE CONVERSA---
```

4.3.2 Testes automáticos

Uma outra funcionalidade extra desenvolvida é a possibilidade da efetuar testes automáticos ao sistema, isto é, escrever previamente as frases que se pretende testar. No momento da complicação do sistema, basta utilizar a flag `-t` ou `--test` e indicar o ficheiro que contém as entradas que pretendemos testar. Este deve estar presente na pasta `testing`.
Segue-se um exemplo de um possível ficheiro de teste.

```
Olá
Preciso de informações sobre a SEI. Achas que me podes ajudar?
Quando é que começa a SEI?
Quando é a sessão de abertura?
A inscrição é gratuita?
Acho que está tudo
Adeus
```

Este geraria a seguinte conversa (segue-se a correspondente entrada no ficheiro de log).

```
User: Olá
Bot: Saudações! Como vai isso?
User: Preciso de informações sobre a SEI. Achas que me podes ajudar?
Bot: Estou aqui para isso! De que precisas?
User: Quando é que começa a SEI?
Bot: A SEI realiza-se no início do segundo semestre.
User: Quando é a sessão de abertura?
Bot: 4 fevereiro segunda-feira,10:00,10:30
User: A inscrição é gratuita?
Bot: A SEI é gratuita, não tens que pagar.
```

```
User: Acho que está tudo
Bot: Espero que tenha sido útil!
User: Adeus
Bot: Adeus companheiro!

---FIM DE CONVERSA---
```

Capítulo 5

Testes do sistema

5.1 Instalação e compilação

Para correr o sistema, isto é, poder comunicar em tempo real com os *chatbots*, utiliza-se o seguinte comando:

```
python3 diretor_dsl.py
```

Este leva a que se use o ficheiro default de especificação (`dsl/dsl.txt`). Caso se queira usar outro, basta utilizar a flag `--dsl` seguida do nome do ficheiro. Em alternativa, é possível utilizar frases pré-feitas guardadas em ficheiros de teste presentes na diretoria `testing`, recorrendo à flag `-t` ou `--test` seguida do nome do ficheiro.

Os requisitos mínimos (a nível de *packages*) necessários para a instalação do programa são os que se seguem:

- `nltk:` usado para remover stopwords das frases em certos bots como *bot_faq* e *bot_lista*.
- `antlr:` usado para verificar a correção (tendo em conta a gramática definida) da especificação fornecida.
- `py_translator:` usado para fazer traduções
- `regex:` usado para funções de expressões regulares.

5.2 Estrutura do sistema

O sistema desenvolvido é constituído pelas seguintes pastas e ficheiros:

- Diretorias `bot_csv`, `bot_dicio`, `bot_exp`, `bot_FAQ`, `bot_lista`, `bot_QA`, `bot_tradutor` e `bot_wiki` que contêm o código referente a cada um dos sete tipos de *chatbots* possíveis.
- Diretoria `data` na qual se armazenam as fontes de informação que são utilizadas pelos *chatbots*.
- Diretoria `dsl` na qual se armazenam os ficheiros com as especificações.
- Diretoria `estados` que contém código referente aos estados (*STATES*) dos *chatbots*.
- Diretoria `gramatica` onde se encontra o ficheiro com a gramática da linguagem bem como código gerado pelo `ANTLR` para o seu funcionamento.

- Diretoria `log` onde se encontra o ficheiro `log.txt` com o registo das conversas.
- Diretoria `testing` onde estão guardados os ficheiros de teste.
- Ficheiro de código `diretor_dsl.py`, `altera_estados.py`, `dsl.py`, `get_regras.py` e `util.py`.

5.3 Testes realizados e Resultados

Mostram-se a seguir alguns testes feitos (valores introduzidos) e os respectivos resultados obtidos, bem como o resultado da aplicação do sistema a um caso de estudo particular, nomeadamente, a Semana da Engenharia Informática da Universidade do Minho.

5.3.1 Exemplos de funcionamento do sistema

Para verificar se o funcionamento do sistema era o esperado realizamos vários testes para cada um dos tipos de *chatbots* com diversas fontes de informação, prioridades e combinações de estados. Nesta subsecção iremos examinar exemplos para cada um deles.

Bot de Expressões

Começando pelo tipo `bot_exp` (bot de expressões), temos como exemplo o *chatbot* das saudações e o *chatbot* das despedidas. Este recorre a expressões regulares para identificar certas expressões num dado *input* e devolve uma das respostas que foram definidas como adequadas. Segue-se um exemplo de especificação dos mesmos, excertos das fontes de informação fornecidas e o ficheiro *log* com as suas interações, respetivamente.

```
!5 CREATE bot_exp FROM saudacoes.json
!5 CREATE bot_exp FROM despedidas.json
```

```
{"lista":[
    {
        "expressoes": ["^Bom dia"],
        "resposta": ["Bom dia para ti também!","Bom dia para ti também! Está um dia lindo, de facto!"]
    },

    ...

    {
        "expressoes": ["^Boas","^Saudações","^Ol[aá]","^Hey","^Oi"],
        "resposta": ["Olá! Tudo bem?","Saudações! Como vai isso?","Boas parceiro."]
    }
]}
```

Listing 5.1: Excerto do ficheiro saudacoes.json

```
User: Olá
Bot: Saudações! Como vai isso?
```

```
User: Bom dia!
Bot: Bom dia para ti também! Está um dia lindo, de facto!

---FIM DE CONVERSA---
```

Bot de Listas

Quanto ao tipo `bot_lista`, temos como exemplo o *chatbot* d'*"Os Lusíadas"* e o *chatbot* dos provérbios. Este devolve a linha do seu ficheiro que faça o maior *match* com o input do utilizador. Segue-se um exemplo de especificação dos mesmos, excertos das fontes de informação fornecidas e o ficheiro *log* com as suas interações, respetivamente.

```
!1 CREATE bot_lista FROM lusiadas.txt
!1 CREATE bot_lista FROM proverbios.txt
```

```
a água salobra, na terra seca, é doce
a ambição, assim como a cólera, é muito má conselheira
a ambição cerra o coração
a amenidade no semblante, anuncia a bondade do coração
a apressada pergunta, vagarosa resposta
a aversão é para o coração, o que a prevenção é para o espírito
a beleza está nos olhos de quem vê
a beleza não se põe na mesa, mas eu não como no chão
```

Listing 5.2: Excerto do ficheiro proverbios.txt

```
User: Sabes me dizer se as galinhas tem dentes?
Bot: Quem ovos vende e galinhas não tem, de algum lado eles vêm.
User: Padre Pedro pegra pregos, pregos pegra o padre Pedro.
Bot: Padre sem sacristão, toca o sino com os pés
```

Bot de QA

No que se refere ao tipo `bot_QA`, efetuamos testes com um exemplo referente à história de portugal. Segue-se um exemplo da especificação usada, excertos das fontes de informação fornecidas e o ficheiro *log* com as suas interações, respetivamente.

```
!5 CREATE bot_QA FROM portugal_QA.txt
```

```
Quem;foi;[primeiro,rei,Portugal];D. Afonso Henriques
Quem;foi;[segundo,rei,Portugal];D. Sancho I
```

Listing 5.3: Excerto do ficheiro portugal_QA.txt

```
User: Quem foi o primeiro rei de Portugal?
Bot: D. Afonso Henriques

---FIM DE CONVERSA---
```

Bot Tradutor

Atentando ao `bot_tradutor`, segue-se um exemplo da especificação usada para o seu teste, excertos das fontes de informação fornecidas e o ficheiro *log* com as suas interações, respetivamente.

```
!3 CREATE bot_tradutor FROM dicionario.json
```

```
{
  "batatas": {
    "en": "potatoes"
  },
  "eu": {
    "en": "i",
    "fr": "Je"
  }
}
```

Listing 5.4: Excerto do ficheiro dicionario.json

```
User: Como se diz carro em inglês?
Bot: A tradução de carro é car.
User: em inglês como se diz batatas?
Bot: A tradução de batatas é potatoes.
User: olá diz-se hi em inglês
Bot: Tradução adicionada!

---FIM DE CONVERSA---
```

Bot Wiki

Atentando ao `bot_wiki`, segue-se um exemplo da especificação usada para o seu teste, excertos das fontes de informação fornecidas e o ficheiro *log* com as suas interações, respetivamente.

```
!5 CREATE bot_wiki FROM wiki.json
```

```
...

"A Bela Adormecida": "A Bela Adormecida é um clássico conto de fadas cuja personagem principal é uma princesa que é enfeitiçada por uma maléfica feiticeira por um dedo picado pelo fuso de um tear.",
 "A Bela e a Fera": "A Bela e a Fera ou A Bela e o Monstro é um tradicional conto de fadas francês.",
"A Bela e o Mestre": "A Bela e o Mestre, reality show a ser transmitido pela TVI em Portugal, junta pela primeira vez José Pedro Vasconcelos e Iva Domingues na apresentação.",
"A Cabana": "A Cabana é um livro do escritor canadense William P.",
"A Carta da Terra": "A Carta da Terra é uma declaração de princípios éticos fundamentais para a construção, no século XXI, de uma sociedade global justa, sustentável e pacífica.",

...
```

Listing 5.5: Excerto do ficheiro wiki.json

```
User: O que é Acabamento?
Bot: Acabamento é o resultado de um ato finalizador de um trabalho de construção
civil, mobiliário, vestuário, artes plásticas, impressão offset, serigráfica,
flexográfica, entre outras, fabrico de livros ou outro tipo de atividade.
User: O que é Noite?
Bot: A noite é o período ocorrido durante a rotação da Terra, conhecido como dia em
que não é recebida a luz do Sol, ou seja, aquela determinada região encontra-se na
parte escura do planeta, é o período do dia compreendido entre o pôr e o Nascer
do sol.

---FIM DE CONVERSA---
```

Bot de *CSVs*

Relativamente ao `bot_csv`, iremos de seguida na secção 5.3.3 examinar um exemplo deste género.

Bot de *FAQs*

Relativamente ao `bot_FAQ`, iremos de seguida na secção 5.3.3 examinar um exemplo deste género.

5.3.2 Estados de comportamento

Segue-se um exemplo de um dos testes feitos ao sistema cujo objetivo foi verificar se as mudanças de estado de comportamento dos *chatbots* ocorrem devidamente.

```
User: és burro
Bot: Não, tu é que és!
User: tenho fome
Bot: Não falo mais contigo.
User: Desculpa
Bot: Estás desculpado.
User: Diz-me uma coisa
Bot: O que precisas?

---FIM DE CONVERSA---
```

5.3.3 Caso de estudo: SEI

Para além dos testes que realizamos ao sistema ao longo do seu desenvolvimento, nomeadamente, cada um dos tipos de *chatbots*, estados, prioridades e como se adaptam às fontes de informação fornecidas, optamos pela sua aplicação a um caso de estudo de modo a testar a sua eficiência num situação mais realista. Deste modo, conseguimos-nos aperceberde pequenos ajustes e melhorias a fazer de modo a melhorar o sistema.

O caso de estudo escolhido foi a Semana da Engenharia Informática, um evento organizado pelo Cesium (Centro de Estudantes de Informática da Universidade do Minho) com o objetivo de promover o curso junto dos alunos da universidade, bem como fomentar curiosidade acerca do mesmo, partilhar conhecimento (sob a forma de palestras e *workshops*) e possibilitar um contacto direto entre os alunos e o mundo do trabalho (devido à participação de várias empresas da área).

O primeiro passo tomado consistiu de uma análise ao *website* da SEI [1].

[1] `https://seium.org/`

Figura 5.1: Página principal do *website* da SEI.

Como é possível observar na imagem 5.1 o *website* apresenta oito categorias principais de informação, nomeadamente, *Agenda*, *Oradores*, *Desafios*, *Como chegar*, *Equipa*, *Inscrições*, *Workshops* e *Badges*. Esta informação, para ser utilizada pelo sistema, deve estar num ficheiro com um dos formatos aceites pelo mesmo. No entanto, recordemos que cada tipo de *chatbot* tem associados formatos de informação específicos. Deste modo, foi essencial escolher um dos tipos de *chatbot* para se poder proceder à extração da informação de forma adequada. Assim sendo, optamos pelo `bot_csv` uma vez que permite a distribuição da informação da informação por categorias tal como no acontece no *website*.

A escolha por este tipo de *chatbot* requer a criação de dois ficheiros como fonte de informação, concretamente, um ficheiro *csv* com os dados e um ficheiro *json* com o *schema* dos mesmos. Para o primeiro, optamos pelas colunas *Atividade*, *Tipo*, *Local*, *Dia*, *Início*, *Fim*, *Descrição*, *Oradores* e *Requisitos*.

agenda_SEI.csv

```
1  Atividade;Tipo;Local;Dia;Início;Fim;Descrição;Oradores;Requisitos
2  Sessão de Abertura;Social;CP2-B1;4 fevereiro segunda-feira;10:00;10:30;Sessão inaugural da edição
   deste ano. Será apresentado o calendário e os parceiros da SEI. E, possivelmente, algumas surpresas.
   ;;
3  Apresentação dos desafios;Social;CP2-B1;4 fevereiro segunda-feira;10:30;11:00;Apresentação de todos
   os desafios que irão acontecer durante a semana.;;
4  Data crunching with Tableau;Workshop;DI-0.05;5 fevereiro terça-feira;09:00;11:00;Vamos fazer uma
   pequena introdução da Xpand-IT, seguida do Workshop de Tableau com os respetivos exercícios e uma
   sessão aberta de questões posteriormente.;Carlos Moreira;Computador com Windows ou Mac OS, ou uma
   máquina virtual com um dos SOs anteriores (De preferência com a última versão do Tableau Desktop
   instalado).
5  Rally das Tascas;Social;Largo do Carpe;6 fevereiro quarta-feira;21:00;00:00;O tradicional Rally das
   Tascas, evento social que pretende promover o convívio entre os participantes da SEI.;;
6  MVP (Minimum Viable Product) para produtos com inteligência artificial;Talk;CP2-B1;5 fevereiro
   terça-feira;14:00;15:00;Criar produtos baseados em inteligência artificial é uma tarefa difícil e
   que requer um investimento de tempo e dinheiro alto e muitas vezes com um retorno não imediato. A
   criação de MVPs pode ser a solução para validar o investimento necessário.; André Pimenta;
```

Figura 5.2: Excerto do ficheiro `agenda.csv`

De seguida, criamos o seu *schema* onde declaramos o tipo de cada coluna (*Objeto*, *Temporal*, *Local* ou *Pessoal*) e os seus respetivos sinónimos. Segue-se um excerto do *schema* criado.

agenda_SEI_schema.json

```
1  {
2      "Atividade":{
3          "Tipo" : "Objeto",
4          "Sinonimos" : ["Conferência","Convenção","Reunião","sessões"]
5      },
6      "Tipo":{
7          "Tipo" : "Objeto",
8          "Sinonimos" : ["categoria"]
9      },
10     "Local": {
11         "Tipo": "Local",
12         "Sinonimos" : ["zona","complexo"]
13     },
```

Figura 5.3: Excerto do ficheiro `agenda_SEI_schema.json`.

No entanto, apercebemo-nos que para a representação de certas informações disponibilizadas este formato não é o mais adequado. Concretamente, questões de caráter geral sobre a SEI e não sobre uma atividade em particular (como é o foco do *csv* criado). Temos como exemplo a secção *Como chegar* da página. Assim, o tipo de *chatbot* mais adequado é, sem dúvida, o `bot_FAQ` já que permite o registo de perguntas que se prevê serem de interesse elevado para o utilizador e a sua respetiva resposta. Segue-se um excerto do ficheiro `json` criado para o efeito.

FAQ_SEI.json

```
{"FAQ":[
    {
        "perguntas": ["O que é a SEI'","Do que trata a SEI","O que sabes sobre a SEI?"],
        "resposta": "A SEI é um evento promovido pelo CesiUM cujo objetivo é promover o gosto pela informática juntos dos alunos."
    },
    {
        "perguntas": ["Onde são as incrições da SEI'","Onde me posso increver?"],
        "resposta": "Para te inscreveres basta aceder ao website da SEI."
    },
    {
        "perguntas": ["A inscrição paga-se?","A SEI é gratuita?","A SEI é grátis?","A inscrição para a SEI é gratuita?"],
        "resposta": "A SEI é gratuita, não tens que pagar."
    },
```

Figura 5.4: Excerto do ficheiro `FAQ_SEI.json`.

A fase seguinte consistiu de testes ao sistema de modo a averiguar o seu correto funcionamento e perceber onde poderia ser melhorado. Para tal, definimos a seguinte especificação.

```
!4 CREATE bot_csv WITH agenda_SEI_schema.json FROM agenda_SEI.csv
!1 CREATE bot_lista FROM lusiadas.txt
!1 CREATE bot_lista FROM proverbios.txt
!5 CREATE bot_wiki FROM wiki.json
!5 CREATE bot_QA FROM portugal_QA.txt
!5 CREATE bot_exp FROM saudacoes.json
!5 CREATE bot_exp FROM despedidas.json
!4 CREATE bot_FAQ FROM FAQ_SEI.json
!3 CREATE bot_exp FROM respostas.json
!3 CREATE bot_tradutor FROM dicionario.json

STATES CHATEADO INFORMATIVO
```

De seguida o sistema foi testado extensivamente e os pontos a melhorar identificados e corrigidos. De modo a facilitar este processo, para além do teste manual (introduzir o input manualmente no sistema, observar a resposta do *chatbot* e seguir a conversa uma interação de cada vez) recorremos também à funcionalidade de testes automáticos na qual escrevíamos uma lista de perguntas/frases input que o sistema leria (uma linha de cada vez) e responderia de acordo. O resultado de cada interação é registado no ficheiro de *log* onde pode ser analisado. Segue-se um exemplo de uma conversa onde se pode verificar o funcionamento do sistema de acordo com a especificação definida acima (com algumas das respostas pelos *chatbots* relativos à SEI).

```
User: Olá
Bot: Boas parceiro.
User: Preciso de informações sobre a SEI. Achas que me podes ajudar?
Bot: Estou aqui para isso! De que precisas?
User: Quando é que começa a SEI?
Bot: A SEI realiza-se no início do segundo semestre.
User: Quando é a sessão de abertura?
Bot: 4 fevereiro segunda-feira,10:00,10:30
User: A inscrição é gratuita?
```

```
10      Bot: A SEI é gratuita, não tens que pagar.
11      User: Acho que está tudo
12      Bot: Espero que tenha sido útil!
13      User: Obrigado, és muito sábio.
14      Bot: De sábio e de louco todos temos um pouco.
15      User: Isso é muito triste...
16      Bot: Triste ventura e negro fado os chama
17      User: Bem, tenho de me ir embora.
18      Bot: Adeus parceiro!
19
20      ——FIM DE CONVERSA——
```

Na conversa acima é possível observar respostas de vários *chatbots* diferentes. Começando pela linha `1` (*"Olá"*), o sistema reconhece esta expressão como sendo uma saudação e sorteia uma das respostas associadas, no caso, *"Boas parceiro."*. Esta ação ocorre devido ao *chatbot* das saudações ativado na especificação com a linha "`!5 CREATE bot_exp FROM saudacoes.json`". Este tem o tipo `bot_exp` (bot de expressões) e a prioridade máxima para que a este tipo de frases se responda sempre com uma saudação adequada e não com, por exemplo, um provérbio, mesmo que o match de palavras seja elevado.

Segue-se a linha `3` com um pedido de informação acerca da SEI (*"Preciso de informações sobre a SEI. Achas que me podes ajudar?"*). Esta frase ativa o estado de comportamento `INFORMATIVO`, uma vez que a pergunta contém uma das expressões responsáveis por esta troca de estados. Como resposta é devolvida a pré-definida para a expressão. Esta situação está prevista no seguinte excerto do código (em `util.py`):

```
1 # estado antes, estado depois, exp regular ativadora, resposta
2 respostas_alteracao_estados = [
3     ('CHATEADO','NORMAL',r'Desculpa','Estás desculpado.'),
4     ('NORMAL','CHATEADO',r'.*','Não, tu é que és!'),
5     ('NORMAL','INFORMATIVO',r'Diz me uma coisa','O que precisas?'),
6     ('NORMAL','INFORMATIVO',r'preciso de informações .*','Estou aqui para isso! De que
      precisas?'),
7     ('INFORMATIVO','NORMAL',r'acho que está tudo','Espero que tenha sido útil!')
8
9 ]
```

Listing 5.6: Excerto do código de `util.py` referente às trocas de estados.

Prosseguindo com a análise, nas linhas `5` e `9` temos duas perguntas sobre a SEI (*"Quando é que começa a SEI?"* e *"A inscrição é gratuita?"* respetivamente). As respostas a estes inputs são dadas pelo *chatbot* do tipo `bot_FAQ` já que estas são duas das *Frequently Asked Questions* a que ele está habilitado a responder (estão definidas na sua fonte de informação `FAQ_SEI.json`).

Na linha `7` (*"Quando é a sessão de abertura?"*) o sistema recorre ao `bot_csv` da SEI para dar resposta, isto é, de entre as respostas recebidas dos vários *chatbots* ativos o sistema escolhe a deste uma vez que como ele efetivamente possui a informação necessária para responder, a sua proposta de resposta vai, tal como esperado, possuir o valor mais elevado de confiança e, portanto, será a devolvida ao utilizador.

O processo de obtenção de resposta é o que segue. Primeiramente, o sistema analisa a pergunta e deteta o tipo da questão ("Quando"), nomeadamente, Temporal. Deteta ainda um elemento presente no `csv` e a sua coluna, concretamente, *"sessão de abertura"* que é uma *"Atividade"*. Deste modo, irá devolver os elementos da linha desta atividade que sejam do tipo temporal, tal como pedido pelo utilizador. Isto é possível recorrendo ao *schema* do csv ao qual tem acesso, pois nele estão definidos o tipo de cada coluna. Assim, à pergunta de `Quando é a atividade X`, o sistema devolve o dia, hora de início e hora de fim da mesma (colunas do tipo Temporal, tal como a questão).

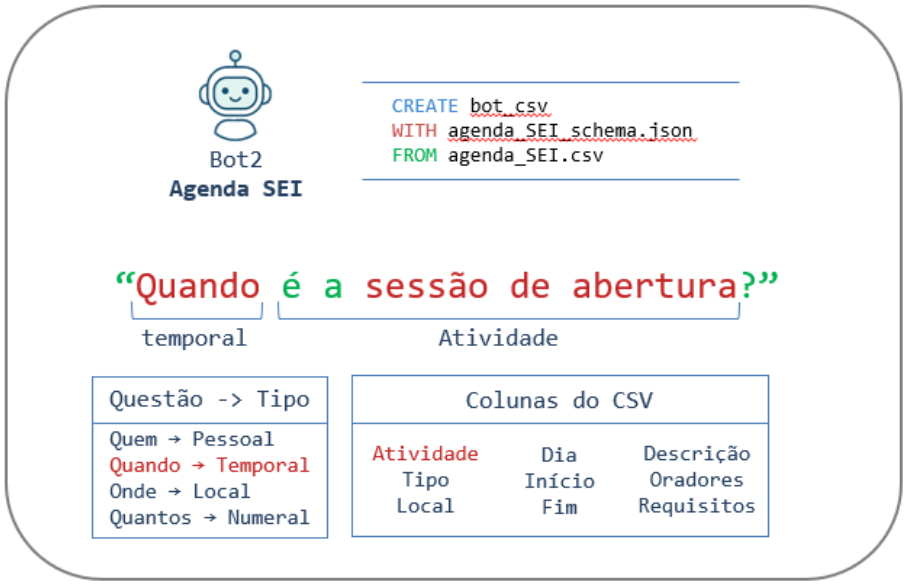

Figura 5.5: Esquema da análise da pergunta.

Figura 5.6: Esquema da seleção dos dados para a resposta.

De seguida, na linha `11` temos à semelhança da linha `3` uma expressão (*"Acho que está tudo"*) que provoca a alteração de estados. No entanto, neste caso a troca é do estado `INFORMATIVO` para o estado `NORMAL`. A regra para esta troca encontra-se detalhada na Listing 5.1., sendo que o *chatbot* responde com a frase nela definida *"Espero que tenha sido útil!"*.

As duas interações seguintes demonstram a capacidade de adaptação do sistema a mudanças de contexto ao longo de uma conversa. Neste caso, a frases que não representam perguntas ou pedidos por parte do utilizador, o sistema responde com *chatbots* de caráter voltado para o entretenimento com o propósito de

divertir e tornar a conversa mais apelativa, o que poderá aumentar o interesse do utilizador pois sendo respostas inesperadas este ficará possivelmente curioso e prolongará a conversa (o que seria menos provável acontecer caso as respostas não contivessem conteúdo novo ou diferente do esperado). Analisando cada umas das interações, primeiramente à frase *"Obrigado, és muito sábio"* segue a resposta *"De sábio e de louco temos todos temos um pouco."*, dada pelo `bot_lista` dos provérbios. De seguida, a *"Isso é muito triste..."* foi respondido *"Triste ventura e negro fado os chama"*, um verso d'*Os Lusíadas* (devido ao `bot_lista` que possui esta obra como fonte de informação).

Por fim, o utilizador despede-se *"Bem, tenho de me ir embora"* e recebe como resposta *"Adeus parceiro"*. À semelhança das saudações, o sistema reconhece a frase como sendo uma despedida e devolve uma das frases adequada para a mesma. A frase da especificação responsável por este *chatbot* é: "`!5 CREATE bot_exp FROM despedidas.json`".

Capítulo 6

Conclusão

O projeto detalhado neste relatório consiste de um sistema que recebendo uma especificação (seguindo as regras de uma *Domain Specific Language* por nós definida) e dada uma fonte de informação, gera um ou ou mais *chatbots* capazes de manter um diálogo, prestar serviços informativos, entre outras aplicações. Neste documento encontra-se detalhada a análise e especificação do problema, o processo de conceção da sua resolução e de desenvolvimento do sistema, bem como os testes que a ele foram efetuados.

O sistema foi desenvolvido com sucesso, tendo sido completadas todas as tarefas necessárias ao bom funcionamento deste, bem como ainda criadas funcionalidades extra no sentido de o valorizar. Assim, este conta com sete tipos diferentes de *chatbots* pré-definidos, três estados de comportamento, diversos formatos de informação aceites e duas *features* adicionais.

A nível de dificuldades sentidas que tivemos de ultrapassar, destacamos a conceção da solução no que toca a projetar um sistema que fosse capaz de juntar de forma inteligente as respostas dos vários *chatbots* e optar pela mais adequada. Ressaltamos ainda a complexidade de tentar humanizar os *chatbots* o mais possível de modo a que melhorar ao máximo a experiência do utilizador.

Por fim, no que diz respeito a trabalho futuro, gostaríamos de implementar mais mecanismos de guardar contexto como é o caso de perfis de utilizadores, isto é, registar informação sobre um dado utilizador como por exemplo o seu género (e.g. para o uso adequado de pronomes), as suas preferências (e.g. para oferecer sugestões adequadas ao mesmo) entre outras.

Apêndice A

Código do Programa

Neste capítulo é possível analisar o código dos ficheiros que consideramos mais relevantes à compreensão do programa.

A.1 Código do ficheiro diretor_dsl.py

```
import random,sys,re,getopt
from util import *
from operator import itemgetter
from math import trunc
from dsl import *
from bot_lista import bot_lista
from get_regras import regras_estado_resposta
from altera_estados import altera_estados

global state_atual
state_atual = 'NORMAL'

# printa uma despedida e escreve o resultado no ficheiro log assim como o fim de conversa
def get_despedida_e_escreve_log():
    despedida = random.choice(despedidas)
    save_log(despedida,'bot')
    print(despedida)
    save_log('','')

# append de uma mensagem ao ficheiro de log
def save_log(msg,ident):
    file = 'log/log.txt'
    file = open(file,'a')
    if ident == 'user':
        file.write('User: '+msg+'\n')
    elif ident == 'bot':
        file.write('Bot: '+msg+'\n')
    else:
        file.write('\n---FIM DE CONVERSA---\n\n')
    file.close()

def responde_test(ficheiro,tuplos,estados):
    # path ficheiro teste
    path_teste = os.getcwd() + '/testing/' + ficheiro
    file = open(path_teste).read()
```

```
    inputs = file.split('\n')
    for input_utilizador in inputs:
        save_log(input_utilizador,'user')
        resposta = responde(input_utilizador,tuplos,estados)
        save_log(resposta,'bot')
    save_log('','')

def responde(input_utilizador,tuplos,estados):
    lista_respostas = []
    global state_atual
    state_anterior = state_atual
    state_atual = altera_estados(input_utilizador,estados,state_atual)

    # para casos especiais em que há alteração de estado
    if state_anterior != state_atual:
        for state_antes,state_depois,regra,resposta in respostas_alteracao_estados:
            if state_antes == state_anterior and state_depois == state_atual:
                match = re.match(regra,input_utilizador,re.IGNORECASE)
                if match is not None:
                    return resposta
        return random.choice(clueless)
    else:
        for regras,dataset,bot,prioridade_bot in tuplos:
            for lista_estados_validos,prioridade_regra,regra,funcao, in regras:
                if state_atual in lista_estados_validos: # só acontece se o estado atual for
     permitido na regra
                    match = re.match(regra,input_utilizador)
                    if match is not None:
                        if callable(funcao):
                            try:
                                resposta,ratio = funcao(match,dataset)
                                if resposta is not None:
                                    if ratio > 1: ratio = 1
                                    confianca = trunc((prioridade_bot + prioridade_regra) *
    ratio)
                                    tuplo = tuple((resposta,confianca,bot))
                                    lista_respostas.append(tuplo)
                            except:
                                pass
        lista_respostas.sort(key=itemgetter(1),reverse=True)
        if lista_respostas:
            return lista_respostas[0][0]
        else:
            for estado_check,funcao_estado in regras_estado_resposta:
                if state_atual == estado_check:
                    resposta = funcao_estado()
                    return resposta
        return random.choice(clueless)

def conversa(tuplos,estados):
    while(True):
        try:
            input_utilizador = input('> ')
            save_log(input_utilizador,'user')
            resposta = responde(input_utilizador,tuplos,estados)
            save_log(resposta,'bot')
            print(resposta)
        except KeyboardInterrupt:
            print('\n')
```

```
            get_despedida_e_escreve_log()
            sys.exit()

def main():
    opts, args = getopt.getopt(sys.argv[1:], 't:', ['test=','dsl='])

    dsl_path = os.getcwd() + '/dsl/'
    dsl_file = dsl_path + 'dsl.txt'
    for opt,arg in opts:
      if opt=='--dsl':
        dsl_file = dsl_path + arg

    if not validate_dsl(dsl_file):
        sys.exit() # dsl nao validada

    triplos_dsl,estados = read_dsl(dsl_file)
    tuplos = change_tuplos(triplos_dsl)

    for opt,arg in opts:
        # print(opt,arg) # debug
        if opt=='-t' or opt=='--test':
            responde_test(arg,tuplos,estados)
            sys.exit()

    conversa(tuplos,estados)

main()
```

Listing A.1: Código do ficheiro diretor_dsl.py

A.2 Código do ficheiro dsl.py

```
import sys, subprocess
import re
from operator import itemgetter
from get_regras import get_regras

# lê a dsl e retorna uma lista de tuplos que contém os bots e o dataset a ser usado
def read_dsl(ficheiro):
    triplos = []
    estados = []
    content = open(ficheiro).read().split('\n')
    for linha in content:
        linha = linha.split(' ')
        if 'STATES'==linha[0]:
            estados = linha[1:]
        elif re.match(r'!([0-5])',linha[0]):
            prioridade_bot = int(linha[0][1])
            datasets = []
            for i in range(1,len(linha)-1):
                if linha[i] == 'CREATE':
                    bot = linha[i+1]
                if linha[i] == 'WITH':
                    datasets.append(linha[i+1])
                if linha[i] == 'FROM':
                    datasets.append(linha[i+1])
        if datasets != [] and bot != []:
            triplo = tuple((bot,datasets,prioridade_bot))
            triplos.append(triplo)
```

```
28      return triplos,estados
29
30  def validate_dsl(ficheiro):
31      MyOut = subprocess.Popen(['python3','gramatica/gramatica.py',ficheiro],
32              stdout=subprocess.PIPE,
33              stderr=subprocess.STDOUT)
34      stdout,stderr = MyOut.communicate()
35      stdout_decode = stdout.decode('utf-8')
36      if(stdout_decode is not ''):
37          print("DSL está mal estruturada. Por Favor corrige-a para continuar.\n")
38          print("Erro:")
39          print(stdout_decode)
40          return False
41      else:
42          print("DSL compilada corretamente.\n")
43          return True
44
45  def change_tuplos(triplos_dsl):
46      tuplos = []
47      for bot,dataset,prioridade_bot in triplos_dsl:
48          regras = get_regras(bot)
49          tuplo = tuple((regras,dataset,bot,prioridade_bot))
50          tuplos.append(tuplo)
51      return tuplos
52
53
54  if __name__ == "__main__":
55      read_dsl(sys.argv[1])
```

Listing A.2: Código do ficheiro dsl.py

A.3 Código do ficheiro get_regras.py

```
1  from bot_lista import bot_lista
2  from bot_tradutor import bot_tradutor
3  from bot_wiki import bot_wiki
4  from bot_csv import bot_csv
5  from bot_QA import bot_QA
6  from bot_exp import bot_exp
7  from bot_FAQ import bot_FAQ
8  from estados import chateado
9  from util import *
10
11 regras_bot_lista = [
12     (['NORMAL'],1,r'(.+)', lambda x,dataset: bot_lista.gera_resposta_dsl(x.group(1),dataset
       [0]))
13 ]
14
15 regras_bot_wiki = [
16     (['NORMAL','INFORMATIVO'],5,r'[Oo] que sabes sobre (.*)\b\??',lambda x,dataset :
       bot_wiki.gera_resposta_dsl(x.group(1),dataset[0])),
17     (['NORMAL','INFORMATIVO'],5,r'[Oo] que é (.*)\b\??',lambda x,dataset : bot_wiki.
       gera_resposta_dsl(x.group(1),dataset[0])),
18     (['NORMAL','INFORMATIVO'],5,r'[Ff]ala me sobre (.*)\b\??',lambda x,dataset : bot_wiki.
       gera_resposta_dsl(x.group(1),dataset[0])),
19 ]
20
21 regras_bot_tradutor = [
```

```
22      (['NORMAL','INFORMATIVO'],4,r'[Cc]omo se diz ([\w ]+) em (\w+)\b\??', lambda x,dataset:
        bot_tradutor.traduz(x.group(1),x.group(2),dataset[0])),
23      (['NORMAL','INFORMATIVO'],5,r'[Qq]ual é a tradução de ([\w ]+) em (\w+)\b\??', lambda x,
        dataset: bot_tradutor.traduz(x.group(1),x.group(2),dataset[0])),
24      (['NORMAL','INFORMATIVO'],4,r'([\w ]+) como se diz em (\w+)\b\??', lambda x,dataset:
        bot_tradutor.traduz(x.group(1),x.group(2),dataset[0])),
25      (['NORMAL','INFORMATIVO'],4,r'[Ee]m (\w+) como se diz (\w+)\b\??', lambda x,dataset:
        bot_tradutor.traduz(x.group(2),x.group(1),dataset[0])),
26      (['NORMAL','INFORMATIVO'],4,r'([\w ]+) em (\w+) diz-se ([\w ]+)\b', lambda x,dataset:
        bot_tradutor.guardar_dicionario(x.group(1),x.group(2),x.group(3),dataset[0])),
27      (['NORMAL','INFORMATIVO'],4,r'([\w ]+) diz-se ([\w ]+) em (\w+)\b', lambda x,dataset:
        bot_tradutor.guardar_dicionario(x.group(1),x.group(3),x.group(2),dataset[0]))
28  ]
29
30  regras_bot_csv = [
31      (['NORMAL','INFORMATIVO'],3,r'(.*)', lambda x,dataset: bot_csv.responde(x.group(0),
        dataset[0],dataset[1])),
32  ]
33
34  regras_bot_QA = [
35      (['NORMAL','INFORMATIVO'],2,r'(.*)', lambda x,dataset: bot_QA.responde(x.group(0),
        dataset[0])),
36  ]
37
38  regras_bot_FAQ = [
39      (['NORMAL','INFORMATIVO'],3,r'(.*)', lambda x,dataset:bot_FAQ.responde(x.group(0),
        dataset[0]) )
40  ]
41
42  regras_bot_exp = [
43      (['NORMAL'],5,r'(.*)', lambda x,dataset: bot_exp.responde(x.group(0),dataset[0]))
44  ]
45
46  regras_estado = [
47      ('CHATEADO',r'Es mesmo ('+insultos_exp_reg+r')', lambda : 'CHATEADO'),
48      ('CHATEADO',r'Es ('+insultos_exp_reg+r')', lambda : 'CHATEADO'),
49      ('CHATEADO',r'Tu és ('+insultos_exp_reg+r')', lambda : 'CHATEADO'),
50      ('CHATEADO',r'Desculpa', lambda : 'NORMAL'),
51      ('INFORMATIVO',r'Diz me uma coisa', lambda:'INFORMATIVO'),
52      ('INFORMATIVO',r'preciso de informações sobre', lambda:'INFORMATIVO'),
53      ('INFORMATIVO',r'preciso de informações da', lambda:'INFORMATIVO'),
54      ('INFORMATIVO',r'acho que está tudo', lambda:'NORMAL'),
55
56  ]
57
58  regras_estado_resposta = [
59      ('CHATEADO',lambda : chateado.responde())
60  ]
61
62
63  def get_regras(bot):
64      regras = []
65
66      if bot == 'bot_lista':
67          regras = regras_bot_lista
68      elif bot == 'bot_tradutor':
69          regras = regras_bot_tradutor
70      elif bot == 'bot_wiki':
71          regras = regras_bot_wiki
```

```
    elif bot == 'bot_csv':
        regras = regras_bot_csv
    elif bot == 'bot_QA':
        regras = regras_bot_QA
    elif bot == 'bot_exp':
        regras = regras_bot_exp
    elif bot == 'bot_FAQ':
        regras = regras_bot_FAQ
    return regras
```

Listing A.3: Código do ficheiro get_regras.py

A.4 Código do ficheiro util.py

```
import os

path_insultos = os.getcwd() + '/data/insultos.txt'
insultos = open(path_insultos).read().split('\n')
insultos_exp_reg = '|'.join(insultos)

clueless = [
    "Não estou a perceber nada...", "Fala-me português!", "O quê?", "Tens a certeza que sabes falar português?!",
    "Andaste a beber?!", "Estás a gozar comigo?!", "Repete lá isso de forma que eu entenda."
]

# estado antes, estado depois, exp reg, resposta
respostas_alteracao_estados = [
    ('CHATEADO','NORMAL',r'Desculpa','Estás desculpado.'),
    ('NORMAL','CHATEADO',r'.*','Não, tu é que és!'),
    ('NORMAL','INFORMATIVO',r'Diz me uma coisa','O que precisas?'),
    ('NORMAL','INFORMATIVO',r'preciso de informações .*','Estou aqui para isso! De que precisas?'),
    ('INFORMATIVO','NORMAL',r'acho que está tudo','Espero que tenha sido útil!')

]
```

Listing A.4: Código do ficheiro util.py

A.5 Código do ficheiro bot_csv.py

```
import csv
import re
import json
import os

# ##################################################################################################

# Variáveis globais/funções/listas

# tipos correspondentes de cada questão
tuplo_questao_tipo = [
    ('quem','Pessoal'),
    ('quando','Temporal'),
    ('onde','Local'),
    ('quantos','Numeral'),
    ('qual','Objeto'),
```

```
    ('em que','Objeto'),
    ('quais','Objeto'),
    ('o que','Objeto')
]

questoes = ['quem','quantos','onde','quando','qual','em que','quais','o que']
questoes_exp_reg = '|'.join(questoes)

lista_exp_reg = [
    # para as situações em que não existe um nome da coluna na frase
    (r'^('+questoes_exp_reg+r')(.*)\b\??'), # começa com questao
    (r'(.*)('+questoes_exp_reg+r')\b\??$'), # acaba com questao
    (r'(.+)('+questoes_exp_reg+r')(.+)\b\??'), # tem a questao a meio
]


# ##########################################################################################

# funções para carregar o ficheiro json e o csv

# faz o open do json e guarda
def save_json(path_files_csv):
    json_file = json.loads(open(path_files_csv).read())
    return json_file

# retorna a lista com as colunas do csv
def get_lista_nomeColunas(json_file):
    dict_keys_nomeColuna = json_file['individuals.csv'].keys()
    lista_nomeColunas = []
    for nome_coluna in dict_keys_nomeColuna:
        lista_nomeColunas.append(nome_coluna)
    return lista_nomeColunas

# retorna  o conteúdo do csv
def openCSV(path):
    with open(path, newline='') as csvfile:
        spamreader = csv.reader(csvfile, delimiter=';')
        valores_csv = []
        lista_nomeColunas = []
        flag = 0
        for row in spamreader:
            if flag == 0:
                flag = 1
                lista_nomeColunas = row
            else:
                valores_csv.append(row)
    return lista_nomeColunas,valores_csv

# ##########################################################################################


# retorna o conteudo do json e do csv
def get_schema_nomeColunas_valorescsv(schema,csv):

    # paths só para testing individual
    # path_csv = os.path.dirname(os.getcwd()) + '/data/' + csv
    # path_schema = os.path.dirname(os.getcwd()) + '/data/' + schema
```

```
71
72      # path geral
73      path_csv = os.getcwd() + '/data/' + csv
74      path_schema = os.getcwd() + '/data/' + schema
75
76      # guarda o conteudo do ficheiro json com o schema
77      schema = save_json(path_schema)
78      (colunas, valores_csv) = openCSV(path_csv)
79
80      return schema,colunas,valores_csv
81
82  # percorre a lista de exp reg e retorna a questao e o resto da frase
83  def procura_mensagem(mensagem):
84      for exp_reg in lista_exp_reg:
85          match = re.search(exp_reg, mensagem,re.IGNORECASE)
86          if match is not None:
87                  num_matches = len(match.groups())
88                  if num_matches == 2:
89                      if match.group(1).lower() in questoes:
90                          questao = match.group(1)
91                          resto_frase = match.group(2)
92                      else:
93                          questao = match.group(2)
94                          resto_frase = match.group(1)
95                  else:
96                      questao = match.group(2)
97                      resto_frase = match.group(1) + match.group(3)
98                  return questao,resto_frase
99      return "",""
100
101 # verifica se o nome da coluna está contida na mensagem
102 def verifica_existe_coluna(resto_frase,colunas):
103     coluna_obj = ""
104
105     nome_colunas_exp_reg = '|'.join(colunas)
106     match = re.search(r'('+nome_colunas_exp_reg+r')',resto_frase,re.IGNORECASE)
107
108     if match is not None:
109         coluna_obj = match.group(1)
110
111     return coluna_obj
112
113 # retorna qual é o tipo de uma questao
114 def busca_tipo_questao(questao):
115     for quest,tipo in tuplo_questao_tipo:
116         if questao.lower() == quest:
117             return tipo
118
119 # verifica se o tipo da coluna objetivo coincide com o tipo da questao
120 def verifica_tipo_nome_coluna(coluna_obj,questao,schema):
121     tipo_questao = busca_tipo_questao(questao)
122     tipo_coluna_obj = schema[coluna_obj.capitalize()]['Tipo']
123     return tipo_questao,tipo_coluna_obj
124
125 # descobre qual é a coluna obj quando não está presente na questao
126 def busca_colunas_tipo_especifico(schema,tipo_questao,colunas):
127     lista_colunas_obj = []
128     for nome_coluna in colunas:
129         tipo_coluna =       sinomimos = schema[nome_coluna]['Tipo']
```

```
130         if tipo_questao == tipo_coluna:
131             lista_colunas_obj.append(nome_coluna)
132     return lista_colunas_obj
133 
134 # vai buscar os sinonimos no schema e cria uma lista de listas com os sinonimos para cada
    coluna
135 def busca_sinonimos_schema(schema,colunas):
136     tuplos_coluna_sinonimos = []
137     for nome_coluna in colunas:
138         sinonimos = schema[nome_coluna]['Sinonimos']
139         tuplo = tuple((nome_coluna,sinonimos))
140         tuplos_coluna_sinonimos.append(tuplo)
141     return tuplos_coluna_sinonimos
142 
143 # verifica se na mensagem contém algum dos sinónimos, caso existe retorna a que coluna
    correspondente
144 # d essa a coluna objetivo
145 def verifica_sinonimos_mensagem(tuplos_coluna_sinonimos,resto_frase):
146     for coluna,sinonimos in tuplos_coluna_sinonimos:
147         sinonimos_exp_reg = '|'.join(sinonimos)
148         match = re.search(r'('+sinonimos_exp_reg+r')',resto_frase,re.IGNORECASE)
149         if match is not None:
150             if match.group(1) is not "":
151                 return coluna
152     return ""
153 
154 # retorna as linhas em que tem o elemento contido na memsangem
155 def procura_elementos(resto_frase,valores_csv):
156     lista_linhas = []
157     linha = 0
158     for row in valores_csv:
159         for elemento in row:
160             if elemento is not "":
161                 if elemento.lower() in resto_frase.lower():
162                     lista_linhas.append(linha)
163         linha += 1
164     return lista_linhas
165 
166 # trata do resultado para o caso em que há várias respostas
167 def trata_resultado(lista_respostas):
168     respostas = []
169     for resposta_linha in lista_respostas:
170         r = ','.join(resposta_linha)
171         respostas.append(r)
172     resposta = '\n'.join(respostas)
173     return resposta
174 
175 # responde quando tem a coluna objetivo
176 def respond_full_agr(colunas,coluna_obj,resto_frase,valores_csv):
177     lista_respostas = []
178     coluna = colunas.index(coluna_obj.capitalize())
179     linhas = procura_elementos(resto_frase,valores_csv)
180     for linha in linhas:
181         # print('coluna: '+ str(coluna) + ' linha: ' + str(linha))
182         resposta = valores_csv[linha][coluna]
183         lista_respostas.append(resposta)
184     return lista_respostas
185 
186 # responde para quando não tem a coluna objetivo
```

```
187 def respond_missing_arg(colunas,lista_colunas_obj,resto_frase,valores_csv):
188     lista_colunas = []
189     lista_respostas = []
190     for coluna_obj in lista_colunas_obj:
191         coluna = colunas.index(coluna_obj.capitalize())
192         lista_colunas.append(coluna)
193     lista_linhas = procura_elementos(resto_frase,valores_csv)
194     # print('lista_colunas: '+ str(lista_colunas) + ' lista_linhas: ' + str(lista_linhas))
195     for linha in lista_linhas:
196         resposta_linha = []
197         for coluna in lista_colunas:
198             resposta = valores_csv[linha][coluna]
199             resposta_linha.append(resposta)
200         lista_respostas.append(resposta_linha)
201     return lista_respostas
202 
203 def responde(mensagem,schema,csv):
204     resposta = ""
205     ratio = 1
206     # retorna o schema e o conteudo do csv
207     (schema,colunas,valores_csv) = get_schema_nomeColunas_valorescsv(schema,csv)
208 
209     # divide a questao do resto da frase e retorna os valores
210     questao,resto_frase = procura_mensagem(mensagem)
211 
212     # vai verificar se a coluna está especifica na mensagem
213     coluna_obj = verifica_existe_coluna(resto_frase,colunas)
214 
215     # tuplo com o nome da colunas e os respetivos sinonimos do schema
216     tuplos_coluna_sinonimos = busca_sinonimos_schema(schema,colunas)
217 
218     if coluna_obj is "":
219         # verificar se por acaso não existe algum sinonimo das colunas na frase
220         coluna_obj = verifica_sinonimos_mensagem(tuplos_coluna_sinonimos,resto_frase)
221 
222     # vai descobrir qual é o tipo da questao e o tipo da coluna_obj
223     if coluna_obj:
224         tipo_questao,tipo_coluna_obj = verifica_tipo_nome_coluna(coluna_obj,questao,schema)
225 
226     # caso a coluna esteja especificada na mensagem
227     if coluna_obj is not "" and not(tipo_questao is not 'Objeto' and tipo_questao !=
    tipo_coluna_obj):
228         lista_respostas = respond_full_agr(colunas,coluna_obj,resto_frase,valores_csv)
229         resposta = (',').join(lista_respostas)
230     # caso a coluna não esteja espeficiada na mensagem
231     else:
232         tipo_questao = busca_tipo_questao(questao)
233         lista_colunas_obj = busca_colunas_tipo_especifico(schema,tipo_questao,colunas)
234         lista_respostas = respond_missing_arg(colunas,lista_colunas_obj,resto_frase,
    valores_csv)
235         resposta = trata_resultado(lista_respostas)
236 
237     if resposta == "":
238         resposta = None
239 
240     return resposta,ratio
```

Listing A.5: Código do ficheiro bot_csv.py

A.6 Código do ficheiro bot_exp.py

```
import os,json
import regex as re
from random import choice

# faz o open do json e guarda
def save_json(path_file):
    json_file = json.loads(open(path_file).read())
    return json_file

# retorna o conteudo do json
def busca_file(nome_file):
    # paths só para testing individual
    # path_file = os.path.dirname(os.getcwd()) + '/data/' + nome_file

    # path geral
    path_file = os.getcwd() + '/data/' + nome_file

    # guarda o conteudo do ficheiro json com o schema
    faq = save_json(path_file)
    return faq

def procura_mensagem(mensagem,expressoes):
    result = 0
    for exp_reg in expressoes:
        match = re.search(exp_reg,mensagem,re.IGNORECASE)
        if match is not None:
            return True
    return False


def responde(mensagem,nome_file):
    ratio = 1
    conteudo = busca_file(nome_file)
    lista_exp = conteudo['lista']

    for exp in lista_exp:
        expressoes = exp['expressoes']
        resposta = exp['resposta']

        if procura_mensagem(mensagem,expressoes) is True:
            return choice(resposta),ratio

# resposta = responde('Hey parceiro','saudacoes.json')
# print(resposta)
```

Listing A.6: Código do ficheiro bot_exp.py

A.7 Código do ficheiro bot_FAQ.py

```
import os,json
import nltk
import regex as re
from random import choice

# faz o open do json e guarda
def save_json(path_faq):
```

```
8       json_file = json.loads(open(path_faq).read())
9       return json_file
10
11  # retorna o conteudo do json
12  def busca_faq(nome_faq):
13      path_faq = os.getcwd() + '/data/' + nome_faq
14      # guarda o conteudo do ficheiro json com o schema
15      faq = save_json(path_faq)
16      return faq
17
18  # remover pontuação e meter o texto da mensagem em minusculas
19  def limpa_texto(mensagem):
20      lista_mensagem = nltk.word_tokenize(mensagem.lower())
21      lista_mensagem = [palavra for palavra in lista_mensagem if palavra not in nltk.corpus.
        stopwords.words('portuguese') and not re.match('\p{punct}', palavra)]
22      return lista_mensagem
23
24
25  def conta_keywords(mensagem_limpa,pergunta_limpa):
26      count_keys = 0
27      for palavra in pergunta_limpa:
28          if palavra in mensagem_limpa.lower():
29              count_keys += 1
30      return count_keys
31
32  def calcula_ratio(count_keys,mensagem_limpa):
33      ratio = 0
34      if count_keys != 0:
35          ratio = count_keys/len(mensagem_limpa)
36      return ratio
37
38  def responde(mensagem,nome_faq):
39
40      lista_respostas = []
41      ratio_cmp = 0
42      faq = busca_faq(nome_faq)
43      ratio = 1
44
45      if mensagem is "":
46        return None,ratio
47      mensagem_limpa = limpa_texto(mensagem)
48      mensagem_limpa_str = ' '.join(mensagem_limpa)
49      lista_faq = (faq['FAQ'])
50      for faq in lista_faq:
51          perguntas = faq['perguntas']
52          resposta = faq['resposta']
53          for pergunta in perguntas:
54              pergunta_limpa = limpa_texto(pergunta)
55              count_keys = conta_keywords(mensagem_limpa_str,pergunta_limpa)
56              if count_keys == len(mensagem_limpa):
57                  lista_respostas.append(resposta)
58      return choice(lista_respostas),ratio
```

Listing A.7: Código do ficheiro bot_FAQ.py

A.8 Código do ficheiro bot_lista.py

```
1  import sys
2  import nltk
```

```
import random
import re
import regex as re
import os

# remove stopwords e pontuação da mensagem revcebida com input
def cleanText(mensagem):
    mensagem = nltk.word_tokenize(mensagem.lower())
    mensagem = [palavra for palavra in mensagem if palavra not in nltk.corpus.stopwords.
    words('portuguese') and not re.match('\p{punct}', palavra)]
    return mensagem

# cria uma lista com as respostas toda encotnradas com base na
# mensagem e na lista com o conteudo dos ficheiros
def find_respostas(palavras,dataset):
    listaRespostas = []
    comp = 1
    n_matches = 0
    for l in dataset:
        count = 0
        for pal in palavras:
            if(my_substring(pal,l)):
                count += 1
        if count > comp:
            comp = count
            listaRespostas = []
            listaRespostas.append(l.capitalize())
        elif count == comp:
            listaRespostas.append(l.capitalize())
    n_matches = comp

    return listaRespostas,n_matches

# verfica se uma palavra está contida numa string
def my_substring (pal,l):
    l = l.lower()
    l = l.split()
    for p in l:
        if(pal == p):
            return True
    return False

# gera uma resposta
def gera_resposta_dsl(mensagem,dataset):
    palavras = cleanText(mensagem)
    ratio = 0
    path_dataset = os.getcwd() + '/data/' + dataset
    dataset = open(path_dataset).read()
    dataset = dataset.split('\n')
    listaRespostas,n_matches = find_respostas(palavras,dataset)
    tam_input = len(palavras)
    if tam_input > 0:
        ratio = (n_matches+1)/tam_input
    if listaRespostas:
        return random.choice(listaRespostas),ratio
    else:
        return None,ratio
```

Listing A.8: Código do ficheiro bot_lista.py

A.9 Código do ficheiro bot_QA.py

```
import re
from bot_QA import formas_totalPT
import sys
import os
from operator import itemgetter


def trata_info(dataset):
    path_dataset = os.getcwd() + '/data/' + dataset
    QeA = open(path_dataset).read()
    QeA = QeA.split('\n')
    info = []
    for linha in QeA:
        resposta = linha.split(';')
        resposta = tuple((resposta[0],resposta[1],resposta[2],resposta[3]))
        info.append(resposta)
    return(info)

def trata_keywords(keywords):
    keywords = re.sub(r'[\[|\]]','',keywords)
    keywords = keywords.split(',')
    return keywords

# pega nos quadruplos e ordena por ratio, depois elimina os que nãoi tem o ratio mais elevado
# e depois ordena esses por o valor de raridade retornado a resposta e o racio do primeiro elemento
def trata_quad(lista_quad):
    lista_quad.sort(key=itemgetter(2),reverse=True)

    max_ratio = lista_quad[0][2]
    for n_keywords,value,ratio,resposta in lista_quad:
        if ratio < max_ratio:
            lista_quad.remove(tuple((n_keywords,value,ratio,resposta)))
    lista_quad.sort(key=itemgetter(1))
    return(lista_quad[0][3],lista_quad[0][2])

def responde(input_utilizador,dataset):
    count_compare = 0
    ratio = 0
    max_value = 999999999
    result = None
    lista_quad = []
    info = trata_info(dataset)

    for questao,verbo,keywords,resposta in info: # percorrer os quadruplos
        keywords = trata_keywords(keywords)
        if questao.lower() in input_utilizador.lower() and verbo.lower() in input_utilizador.lower():
            count = 0
            value = 0
            ratio = 0
            for key in keywords: # percorrer as keywords de um quadruplo
                if key.lower() in input_utilizador.lower():
                    count += 1
                    value += formas_totalPT.dicRank.get(key)
            ratio = count / len(keywords)
```

```
55
56              if count > count_compare:
57                  count_compare = count
58                  lista_quad = []
59                  quad = tuple((count,value,ratio,resposta))
60                  lista_quad.append(quad)
61              elif count == count_compare:
62                  quad = tuple((count,value,ratio,resposta))
63                  lista_quad.append(quad)
64      if lista_quad:
65          (resposta,racio) = trata_quad(lista_quad)
66          return resposta,racio
67      else:
68          return None,0
```

Listing A.9: Código do ficheiro bot_QA.py

A.10 Código do ficheiro bot_tradutor.py

```
1  import os, sys, getopt
2  import regex as re
3  import random
4  import json
5  from py_translator import Translator
6
7  from .listaLinguas import *
8
9  ##### Funçoes #####
10 # traduz uma dada palavra para uma dada linguagem
11 def traduz(palavra,linguagem,ficheiro):
12     abrevLinguagem = linguas.get(removeAccents(linguagem).lower())
13     if abrevLinguagem is not None:
14         dict = verifica_dicionario(palavra,abrevLinguagem,ficheiro)
15         cache = verifica_cache(palavra,abrevLinguagem)
16         if dict:
17             traducao = dict
18         elif cache:
19             traducao = cache
20         else:
21             traducao = Translator().translate(palavra,abrevLinguagem).text
22             guardar_cache(palavra,abrevLinguagem,traducao) # guarda nova traduçao em cache
23         resposta = "A tradução de " + palavra + " é " + traducao + "."
24         return resposta,1
25     else: # nao encontrou a lingua
26         return None,0
27
28
29 ### Funcoes auxiliares de CACHE e DICIONARIO ##
30 # guarda em cache a traduçao de uma palavra para um linguagem
31 def guardar_cache(palavra,linguagem,traducao):
32     cache_path = os.getcwd() + "/bot_tradutor/cache.json"
33     cache_json = json.loads(open(cache_path).read())
34     if cache_json.get(palavra):
35         cache_json[palavra].update({linguagem:traducao})
36     else:
37         cache_json.update({palavra:{linguagem:traducao}})
38     f = open(cache_path, "w")
39     prettyJSON = json.dumps(cache_json,sort_keys=True, indent=2)
40     f.write(prettyJSON)
```

```
41
42 # guarda no dicionario pessoal a traduçao de uma palavra para um linguagem
43 def guardar_dicionario(palavra,linguagem,traducao,ficheiro):
44     dicio = ficheiro
45     path_to_data = os.getcwd() + "/data/"
46     path_to_dicio = path_to_data + dicio
47     abrevLinguagem = linguas.get(linguagem.capitalize())
48     dict_json = json.loads(open(path_to_dicio).read())
49     if dict_json.get(palavra):
50         dict_json[palavra].update({abrevLinguagem:traducao})
51     else:
52         dict_json.update({palavra:{abrevLinguagem:traducao}})
53     f = open(path_to_dicio, "w")
54     prettyJSON = json.dumps(dict_json,sort_keys=True, indent=2)
55     f.write(prettyJSON)
56     respostas = ["Tradução adicionada!","Adicionado ao dicionario."]
57     return random.choice(respostas),1
58
59 # procura na cache se já existe a traduçao de uma palavra para uma linguagem
60 def verifica_cache(palavra,linguagem):
61     cache_json = json.loads(open(os.getcwd()+"/bot_tradutor/cache.json").read())
62     if cache_json.get(palavra):
63         if cache_json[palavra].get(linguagem):
64             return cache_json[palavra][linguagem]
65     # se não tiver a palavra na linguagem correta, returna None
66     return None
67
68 # procura no dicionario se já existe a traduçao de uma palavra para uma linguagem
69 def verifica_dicionario(palavra,linguagem,ficheiro):
70     dicio = ficheiro
71     path_to_data = os.getcwd() + "/data/"
72     path_to_dicio = path_to_data + dicio
73     dict_json = json.loads(open(path_to_dicio).read())
74     if dict_json.get(palavra):
75         if dict_json[palavra].get(linguagem):
76             return dict_json[palavra][linguagem]
77     # se não tiver a palavra na linguagem correta, returna None
78     return None
79
80
81 ##### Funcao para falar com o BOT #####
82 # funcao para uso do bot individualmente
83 def talk():
84     while True:
85         mensagem = input('Eu: ')
86         mensagem = re.search(r'(?:.* )?(.+) em (\w+)\b\??', mensagem)
87         if mensagem is not None:
88             palavra = mensagem.group(1)
89             linguagem = mensagem.group(2).capitalize()
90             return traduz(palavra,linguagem)
91         else: # nao deu match a frase
92             return None
93             # return random.choice(matchFailed) # resposta de falha
94
95
96 ##### MAIN #####
97 def main(options):
98     if '-h' in options:
99         print_help()
```

```
    elif '-x' in options:
        talk()

# print de help ao utilizador
def print_help():
    print('BOT_TRADUTOR\n')
    print('    NOTE: If no option is given, nothing will be executed')
    print('\tOPTIONS:')
    print('\t    -h, --help\n\t    \tPrint this message and exit.')
    print('\t    -x, --exec\n\t    \tExecutes the bot.')
    print('')


##### Run #####
if __name__ == "__main__": # corre quando é o ficheiro principal
    try:
        # na listagem de options nao se coloca o - ou --
        short_opts = 'hxr'
        long_opts = ['help','exec','rules']
        options, remainder = getopt.getopt(sys.argv[1:],short_opts,long_opts)
        options = dict(options) # options = [(option, argument)]
        if remainder:
            print('Too many args')
            sys.exit(1)
        else:
            main(options)
    except getopt.GetoptError as err:
        print('ERROR:', err)
        sys.exit(1)
```

Listing A.10: Código do ficheiro bot_tradutor.py

A.11 Código do ficheiro bot_wiki.py

```
import json
import os

def gera_resposta(palavra, lista_DIC):
    for dic in lista_DIC:
        path = "data/" + dic
        palavra = palavra.capitalize()
        wiki = json.loads(open(path).read())
        resposta = wiki.get(palavra)
        return resposta

def gera_resposta_dsl(palavra,dataset):
    ratio = 1
    path_dataset = os.getcwd() + '/data/' + dataset
    palavra = palavra.capitalize()
    wiki = json.load(open(path_dataset))
    resposta = wiki.get(palavra)
    return resposta,ratio
```

Listing A.11: Código do ficheiro bot_wiki.py

Bibliografia

[1] The future of chatbots from the experts. https://www.searchenginejournal.com/future-of-chatbots/278595/. Último acesso em 28/04/2019.

[2] Microsoft ceo satya nadella says chatbots will revolutionize computing. https://www.onmsft.com/news/microsoft-ceo-satya-nadella-says-chatbots-will-revolutionize-computing. Último acesso em 28/04/2019.

[3] Person-centered therapy. https://www.psychologytoday.com/us/therapy-types/person-centered-therapy. Último acesso em 26/04/2019.