

Projet TATIA : Rapport :

Conception et implémentation d'un système de questions réponses en langue naturelle sur des données structurées avec système de reconnaissance vocale intégré

Les systèmes de questions/réponses sont des systèmes qui répondent à une question posée en langage naturel, par l'extraction d'une réponse précise à partir d'un corpus de documents. Le projet que nous avons réalisé ici s'appuie sur base *DBpedia* qui propose une version structurée et normalisée au format du web sémantique des contenus de *Wikipedia*. Le système que nous avons conçu requiert l'utilisation d'un **interpréteur Python** afin de permettre son exécution ainsi que d'une **connexion internet active** afin d'extraire les réponses de la base *DBpedia*. Le programme a en particulier été testé avec Python en **version 3.9 et 3.9.1** mais devrait cependant fonctionner sous des versions plus anciennes ou plus récentes de **Python 3**. De plus un certain nombre de packages Python supplémentaires doivent être installés sur la machine afin de permettre l'exécution du programme. Le nom des packages et les différentes commandes d'installation sont détaillés dans la première section de ce rapport.

Table des matières

Prérequis au lancement du programme

Packages indispensables pour le fonctionnement du projet

L'installation de tkinter est nécessaire à l'affichage de l'interface graphique (gui)

Packages nécessaires aux fonctionnalités vocales

Note concernant l'installation de PyAudio

Choix de Spacy

Fonctionnement global du traitement question / réponse

Configuration du projet

Détail du code par fonctions

Type de questions prises en compte dans notre programme

Interface graphique (gui)

Fonctionnalités vocales

Reconnaissance vocale

Lecture vocale

Evaluation de notre système

Améliorations possibles

Prérequis au lancement du programme

En plus d'un interpréteur Python et d'une connexion internet active, des packages Python supplémentaires sont nécessaires au fonctionnement du code de ce projet. La liste ci-après contient le nom des packages à installer ainsi que la commande permettant son installation avec le gestionnaire de paquet **PIP** accompagné d'une brève description (nous reviendrons en détail sur le rôle concret des éléments apportés par ces packages dans notre projet plus tard dans ce rapport).

Les packages peuvent être installés séparément ou d'un seul coup avec les 2 lignes suivantes :

```
pip install spacy bs4 lxml deep-translator SpeechRecognition gtts pygame PyAudio  
+ python -m spacy download fr_core_news_lg
```

Packages indispensables pour le fonctionnement du projet

- **Spacy :** `pip install spacy`

SpaCy est une bibliothèque logicielle Python libre de traitement automatique des langues. Cette bibliothèque inclut par exemple des outils tels qu'un tokenizer ou un PoS tagger, mais a aussi pour grand avantage d'inclure un NER prenant en charge la langue française.

- **Modèles statistiques pré-entraînés en français pour Spacy :**

```
python -m spacy download fr_core_news_lg (après installation de spacy!)
```

Installation d'un modèle français pré-entraînés pour Spacy permettant la prédiction des entités nommée et la détermination des dépendances syntaxiques. Il s'agit ici d'un réseau neuronal convolutif formé sur UD French Sequoia et WikiNER en français.

Note : un bug interne à numpy version 1.19.4 peut faire planter, dans ce cas revenir à la version 1.19.3 : `pip uninstall numpy et pip install numpy==1.19.3`

- **Beautiful Soup 4:** `pip install bs4`

Bibliothèque Python permettant d'extraire des données de fichiers HTML et XML.

- **Lxml:** `pip install lxml`

Bibliothèque permettant le traitement de XML et HTML dans le langage Python.

- **Deep Translator:** `pip install deep-translator`

Bibliothèque permettant la traduction entre différentes langues de manière simple en utilisant plusieurs traducteurs.

L'installation de tkinter est nécessaire à l'affichage de l'interface graphique (gui)

Tkinter (de l'anglais Tool kit interface) est la bibliothèque graphique libre d'origine pour le langage Python, permettant la création d'interfaces graphiques. Elle vient d'une adaptation de la bibliothèque graphique Tk écrite pour Tcl

Note : le package est normalement inclus nativement avec Python mais si ce n'est pas le cas, exécuter, si nécessaire, `sudo apt-get install python3-tk` sous Linux / Ubuntu.

Packages nécessaires aux fonctionnalités vocales

- **Speech Recognition**: `pip install SpeechRecognition`

Permet la transformation de phrases orales en texte écrit

- **gTTS**: `pip install gtts`

Permet la transformation de texte écrit en fichier mp3

- **Pyglet**: `pip install pyglet`

Permet la lecture de fichier mp3

- **PyAudio**: `pip install PyAudio` `sudo apt-get install python-tk`

Outils assurant divers traitements audios nécessaires au fonctionnement de certaines fonctionnalités de reconnaissance vocale du projet.

Note concernant l'installation de PyAudio

Dans certains cas l'installation de PyAudio avec la commande « *pip* » peut s'avérer capricieuse, en particulier avec des versions récentes de Python et/ou de la commande *pip*. On notera ici deux solutions qui devraient permettre la résolution de ce problème selon le système d'exploitation utilisé.

Avec **Windows** les deux commandes suivantes devraient permettre la bonne installation de PyAudio : `pip install pipwin` suivi de `pipwin install pyaudio`

Sous Linux (**Ubuntu**) les deux commandes suivantes devraient permettre la bonne installation de PyAudio : `sudo apt-get install -y portaudio19-dev` suivi de `pip install PyAudio`

Si une erreur persiste exécuter `sudo apt-get install python3.9-dev` et réessayer d'installer PyAudio.

*Note : Il est possible d'exécuter le programme sans bénéficier des fonctionnalités liées à la reconnaissance ou la lecture vocale. L'installation des packages cités dans cette dernière partie ne sont alors pas nécessaires. Se reporter à la section « **configuration du projet** » plus loin dans ce rapport.*

Choix de Spacy

Le système de questions/réponses que nous avons conçu s'appuie principalement sur les outils apportés par la bibliothèque **Spacy**. Nous avons effectué le choix de Spacy pour plusieurs raisons. Tout d'abord nous souhaitions concevoir notre système en français et Spacy possède des modèles statistiques dans cette langue ainsi qu'un support natif du *Named-Entities Recognition (NER)* pour le **français** ce qui n'est pas le cas de ces principaux concurrents comme NLTK. De plus les algorithmes de tokenization utilisées dans la bibliothèque Spacy sont efficaces et rapides et la bibliothèque utilise une approche orientée objet facilement lisible. De plus certaines fonctionnalités telles qu'un support de la méthode d'apprentissage « *word embedding* » sont présentes avec Spacy alors qu'elles ne sont par exemple pas supportées par NLTK.

Pour permettre de meilleures performances, nous avons fait le choix d'exécuter Spacy sur le GPU de la machine lorsque cela été possible (`spacy.prefer_gpu()`) et d'utiliser le corpus de texte en français pré-entraînés proposé par Spacy : « *fr_core_news_lg* ».

Fonctionnement global du traitement question / réponse

Cette introduction a pour but d'expliquer de manière générale le fonctionnement du programme que nous avons conçu avant d'entrer plus en détail dans le rôle de chaque fonction. Le rôle général du système de question / réponse que nous avons conçu consiste à prendre une question saisie par l'utilisateur en entrée et à afficher une réponse à cette question en sortie.

La 1^{ère} étape consiste donc à extraire de la question posée les éléments qui peuvent nous permettre d'effectuer une requête capable d'afficher la réponse. Pour cela nous nous sommes principalement appuyés sur des méthodes découlant d'une analyse lexicale (**tokenization**), d'un étiquetage morpho-syntaxique (**POS tagging** [part-of-speech tagging]) et sur une reconnaissance d'entités nommées (**NER** [named-entity recognition]). Ces différentes étapes nous permettent, par exemple, d'extraire les mots-clés les plus importants dans la question (fonction **get_hotwords**). Afin d'identifier le type de questions posées et de récupérer les éléments nécessaires pour effectuer notre requête nous nous sommes également appuyés sur l'utilisation d'**expressions régulières associées à l'analyse NER** de la question (**fonction exp_reg**). Cette étape nous permet d'identifier le type de recherche induite par la question (personne, lieu, date, etc) et de conserver les seuls mots nécessaires à l'exécution de la requête.

La 2^{ème} étape consiste à identifier la page dbpedia sur laquelle se trouve la réponse que l'on recherche. Afin de pouvoir identifier précisément le nom de la page recherchée on se base sur le mot-clé principal identifié lors de l'étape précédente dans la question et on utilise l'API **DBpedia Lookup** qui nous renvoie le label précis de la page dbpedia correspondant à ce mot-clé (fonction **lookup_keyword**). L'API nous permet ainsi d'effectuer une requête http renvoyant une page xml associé au mot-clé recherché et grâce à la bibliothèque *BeautifulSoup4* et au parser inclus dans la bibliothèque *Lxml*, on récupère le label exact nous permettant d'accéder à la page dbpedia contenant l'information recherchée. L'API fonctionnant en anglais, on utilise une étape de traduction intermédiaire sur le mot-clé de la question (français vers anglais).

Note importante : *Nous avons malheureusement fait face à un problème inattendu à cette étape ayant fortement impacté le projet sur lequel nous travaillons. En effet nous utilisons depuis le début l'API suivante : <https://lookup.dbpedia.org/api/search/KeywordSearch> (<https://github.com/dbpedia/lookup>) qui semble avoir rencontré de forts dysfonctionnements à partir de fin décembre la rendant inutilisable.*

Proxy Error

The proxy server received an invalid response from an upstream server.
The proxy server could not handle the request

Reason: **Error reading from remote server**

Apache/2.4.38 (Debian) Server at lookup.dbpedia.org Port 443

En conséquence nous avons décidé de changer l'API utilisée :

<http://akswnc7.informatik.uni-leipzig.de/lookup/api/search> (<https://github.com/dbpedia/dbpedia-lookup>)

Le programme continue avec cette nouvelle API de fonctionner globalement de la façon dont nous l'avons conçu mais cela a impacté la dernière ligne droite du projet ainsi que quelques recherches par mot-clé lors de cette étape qui s'effectuaient correctement avec l'API initialement utilisée mais pas avec celle-ci (par exemple le mot-clé « Marseille » de type ville renvoyé bien le label correspondant à la page dbpedia de la ville sur la première API mais pas sur la nouvelle, nous avons cependant implémenté de nouvelles solutions pour régler ou contourner ces problèmes dès que nous en avons constaté).

La 3^{ème} étape consiste à exécuter une requête SPARQL sur la page dbpedia identifiée afin de récupérer la donnée correspondant à la réponse à la question. Pour effectuer cette requête on s'appuie sur la page dbpedia identifiée à l'étape précédente ainsi qu'aux autres mots-clés que l'on avait extrait de la question et qui nous permettent de savoir quelles données extraire de la page dbpedia. La requête elle-même est effectuée en se servant de la fonction « **query** » directement via [l'interface dbpedia](#). La formulation de la requête est effectuée en se servant des mots-clés et de la page précédemment identifiée par les fonctions « **requete_dbpedia** » ou « **requete_dbpedia_multiple** » selon le type de réponse attendue (une seule réponse ou plusieurs réponses [par exemple si on cherche des créateurs pour un site web il peut y en avoir plusieurs mais si on cherche le roi d'un pays il n'y en a qu'un, cela dépend donc du type de la question]).

Enfin la dernière étape consiste simplement à renvoyer le résultat de la requête qui devrait répondre à la question posée initialement par l'utilisateur.

Configuration du projet

Il est laissé à l'utilisateur du code de ce projet la possibilité d'effectuer quelques paramétrages afin d'activer ou de désactiver certaines fonctionnalités. Pour cela l'utilisateur doit se reporter à la fonction « **main** » du code contenu dans le fichier Python (à la fin) et configurer les différents booléens préconfigurés soit sur « **True** » soit sur « **False** » selon la configuration désirée. La liste des différents booléens configurables et des conséquences associées est la suivante :

gui : **True** pour activer l'interface graphique, **False** pour afficher les entrées et sorties dans la **console** de façon purement **textuelle** sans mise en forme

vocal : **True** pour activer les fonctionnalités de reconnaissance et de lecture vocale, **False** pour désactiver toutes fonctionnalités vocales (*Note : Il n'est pas possible d'activer les fonctionnalités vocales dans le cas où l'interface graphique est désactivée*).

exemple_questionsxml : **True** pour afficher et répondre par défaut aux exemples de questions contenues dans le jeu de donnée « *questions.xml* » dès le début du programme sans action de l'utilisateur (lorsque les questions sont toutes traitées, la main est redonnée à l'utilisateur afin qu'il puisse entrer ses propres questions), configurez **False** pour ne pas traiter ce jeu de donnée par défaut.

exemple_autres : Même principe que ci-dessus mais pour des exemples autres que ceux du jeu de donnée « *question.xml* »

Détail du code par fonctions

Le code de ce projet (contenu dans le fichier « projet.py ») est divisé en plusieurs fonctions remplissant différents rôles. Ces fonctions sont réparties en 3 grandes parties. Les fonctions servant au **fonctionnement même du système de questions / réponses**, les fonctions permettant la **gestion de l'interface graphique (GUI)** et enfin les fonctions permettant le **fonctionnement de la reconnaissance et de la lecture vocale**. Nous nous intéresserons ici à la première catégorie, c'est-à-dire toutes les fonctions de base de notre projet implémentant le système de questions / réponses. Nous nous reviendrons plus tard dans ce rapport sur le fonctionnement de l'interface graphique et du système vocal intégré. Les fonctions représentées sur la page suivante sont renseignées dans l'ordre d'exécution, depuis la première fonction appelé dans le « **main** » de notre programme et prenant en entrée la question de l'utilisateur, à la dernière fonction renvoyant la réponse à la question initialement posée.

affichage_reponse(question, gui=True)

Fonction prenant en paramètre la question entrée par l'utilisateur et un booléen indiquant si l'interface graphique est activé (**True**) ou non (**False**). La fonction est principalement responsable d'appeler d'autres fonctions permettant le traitement de la question et la gestion de l'affichage de la réponse selon le mode d'exécution (sur sortie standard ou sortie graphique).



reponse(question)

Fonction prenant en paramètre la question de l'utilisateur, effectuant des prétraitements de base sur cette question (gestion des apostrophes, sauts de lignes etc.) et extrayant les principaux mots-clés de la question. Cette extraction des mots-clés s'appuie sur la fonction **get_hotwords** qui renvoie une liste des principaux mots-clés de la question grâce au traitement **NER** et **PosTagger** possible en français grâce au corpus de texte et à la bibliothèque **Spacy**. Lorsqu'il est possible de compléter l'extraction des données importantes sur la seule base des mots-clés trouvés on passe directement à la fonction requête dbpedia permettant d'obtenir la réponse. Sinon la question est passée à une autre fonction s'appuyant sur les expressions régulières pour extraire les informations de la question.



exp_reg(question)

Fonction prenant en paramètre la question et permettant d'extraire les mots de la question nécessaire afin d'effectuer la requête sparql. Les nombreux patterns que nous avons créés dans cette fonction s'appuie non seulement sur la présence de certains mots spécifiques mais aussi sur l'analyse NER et PosTagger. [Pour plus d'informations sur la syntaxe utilisée se référer à la documentation.](#)



lookup_keyword(requete, type, translate=True)

Fonction prenant en paramètre l'objet que l'on vient d'extraire de la question et le type d'objet que l'on recherche (pour faciliter la recherche). On se sert ensuite de l'API DBpedia Lookup que l'on va consulter pour déterminer l'intitulé exacte de la page DBpedia sur laquelle se trouve l'information que l'on recherche. L'API nous renvoie une page XML que l'on parse à l'aide des packages Python installés précédemment et depuis laquelle on récupère le label qui peut le plus correspondre à l'information que l'on a extraite de la question de l'utilisateur. L'API fonctionnant en anglais, une **traduction intermédiaire** peut être réalisé avant la recherche sur l'API (sauf dans certains cas comme des noms de lieux ou de personne).



requete_dbpedia[multiple] (requete, predicate, entity of type="dbo") get_abstract(requete)

Ces 3 fonctions permettent la recherche de la réponse demandée auprès de la base DBPedia. La requête sparQL auprès de DBPedia est formulé par la fonction **json_load** (qui prend en compte les différents namespaces [dbo, dbp etc.] et adapte la requête en conséquence) puis transmise à la fonction **query** prenant en entrée la requête dont on souhaite recevoir la réponse ainsi que l'adresse vers le serveur où la requête doit être envoyée. L'écriture de la requête se base sur toutes les informations extraites de la question et de son analyse (prédicat et objet de la question après traitement par l'API lookup). La requête DBpedia renvoyant un lien vers la réponse, un second traitement est effectué pour extraire de ce lien la réponse textuelle à la question.

La fonction requete_dbpedia est alors capable de renvoyer la réponse que l'on a déterminé après toutes ces étapes, alors que la version "multiple" de cette fonction permet la même chose mais autorise le renvoi de plusieurs réponses (nécessaire selon le type de question, par exemple "donner tous les acteurs d'un film"). La fonction "get_abstract" se contente elle du renvoi de descriptions **en français si disponible sinon en anglais.**

Type de questions prises en compte dans notre programme

Le tableau suivant donne une liste du type de questions prises en charge par le système que nous avons conçu.
La 2^{ème} colonne indique le type de réponse apportée pour chaque question.


Type de question	Réponse apportée par le système
Quelle est la capitale de <i>nomPays</i> ?	La capitale de <i>nomPays</i>
Quel est l'indicatif téléphonique de <i>nomVille</i> ?	L'indicatif téléphonique de <i>nomVille</i>
Dans quel musée est exposé/présenté <i>nomOeuvreArt</i> ?	Le nom du musée où <i>nomOeuvreArt</i> est exposée
Quel est l'endroit le plus haut de <i>nomLieu</i> ?	L'endroit le plus élevé de <i>nomLieu</i>
Qui est/était l'époux/l'épouse/etc. de <i>nomPersonne</i> ?	Nom de l'époux/l'épouse/etc. de <i>nomPersonne</i>
En quel langage de programmation a été écrit <i>nomLogiciel</i> ?	Nom du langage de programmation avec lequel <i>nomLogiciel</i> a été écrit
Quelle est la monnaie de <i>nomPays</i> ?	Monnaie de <i>nomPays</i>
Quand se déroula <i>nomEvenement</i> ?	La date de <i>nomEvenement</i>
Quelle est la date de naissance de <i>nomPersonne</i> ?	La date de naissance de <i>nomPersonne</i>
Quel est le site web de <i>nomEntreprise</i> ?	Le site web de <i>nomEntreprise</i>
Qui est le maire de <i>nomVille</i> ?	Le nom du maire de <i>nomVille</i>
Qui est le président de <i>nomPays</i> ?	Le nom du président de <i>nomPays</i>
Quels sont les états voisins de <i>nomEtat</i> ?	Les états voisins de <i>nomEtat</i>
Dans quel pays se trouve <i>nomLieu</i> ?	Le pays où se trouve <i>nomLieu</i>
Quels sont les langues officielles de <i>nomPays</i> ?	Les langues de <i>nomPays</i>
A qui appartient <i>nomEntreprise</i> ?	Le propriétaire de <i>nomEntreprise</i>
Quels prix ont été gagné par <i>nomOrganisation</i> ?	Les prix gagnés par <i>nomOrganisation</i>
Qui est le créateur de <i>nomCreation</i> ?	Le nom du/des créateur(s) de <i>nomCreation</i>
Qui a conçu <i>nomConstruction</i> ?	Le nom du concepteur de <i>nomConstruction</i>
Donne-moi tous les acteurs jouant dans le film <i>nomFilm</i>	Les noms des acteurs de <i>nomFilm</i>
Quel est la cause de décès de <i>nomPersonne</i> ?	La cause de décès de <i>nomPersonne</i>
Qui est <i>nomPersonne</i> ?	Description détaillée de <i>nomPersonne</i>
Combien d'employés a <i>nomEntreprise</i> ?	Nombre d'employés embauchés par <i>nomEntreprise</i>
Quelle <i>nomFleuve</i> est traversée par <i>nomPont</i> ?	Le nom du cours d'eau qui traverse <i>nomPont</i>
Quels sont les pays traversés par <i>nomFleuve</i> ?	Les noms des pays qui sont traversés par <i>nomFleuve</i>
Qui a écrit le livre <i>nomLivre</i> ?	Le nom de l'auteur de <i>nomLivre</i>

Note : les réponses renvoyées correspondent aux données présentes dans la base DBpedia et ne sont pas forcément à jour.

Interface graphique (gui)

```
Quelle cours d'eau est traversé par le pont de Brooklyn ?  
East River (New York)  
  
Qui est le créateur de Wikipedia ?  
Larry Sanger et Jimmy Wales  
  
Dans quel pays commence le Nil ?  
Rwanda et Éthiopie  
  
Quel est l'endroit le plus haut du Karakoram ?  
K2  
  
Qui a conçu le pont de Brooklyn ?  
John Augustus Roebling  
  
Qui est le créateur de Goofy ?  
Paul Murry et Walt Disney et Frank Webb (cartoonist)  
  
Qui est le maire de New York City ?  
Bill de Blasio  
  
Quels sont les pays traversés par l'Ienisseï ?  
Mongolie et Russie  
  
Dans quel musée est exposé Le Cri de Munch ?  
None  
Quels sont les états voisins de l'Illinois ?  
Illinois  
Missouri (État) et Indiana et Kentucky et Iowa et Wisconsin  
  
Qui était l'épouse du président américain Lincoln ?  
Mary Todd Lincoln  
  
En quel langage de programmation a été écrit GIMP ?  
GTK+ et C (langage)  
  
Dans quel pays se trouve le lac Limerick ?  
Canada  
  
Quel est la devise de la Tchéquie ?  
Couronne tchèque  
  
Qui a développé World of Warcraft ?  
Blizzard Entertainment
```

Version console classique (gui désactivé)

 Système de questions-réponses en français

Bot: Bonjour ! Posez moi une question, j'essaierai d'y répondre au mieux :)

Vous: Quelle cours d'eau est traversé par le pont de Brooklyn ?
Bot: East River (New York)

Vous: Qui est le créateur de Wikipedia ?
Bot: Larry Sanger et Jimmy Wales

Vous: Dans quel pays commence le Nil ?
Bot: Rwanda et Éthiopie

Vous: Quel est l'endroit le plus haut du Karakoram ?
Bot: K2

Vous: Qui a conçu le pont de Brooklyn ?
Bot: John Augustus Roebling

Vous: Qui est le créateur de Goofy ?
Bot: Paul Murry et Walt Disney et Frank Webb (cartoonist)

Vous: Qui est le maire de New York City ?
Bot: Bill de Blasio


Vous: Quels sont les pays traversés par l'Ienisseï ?
Bot: Mongolie et Russie

Vous: Dans quel musée est exposé Le Cri de Munch ?
Bot: None

Vous: Quels sont les états voisins de l'Illinois ?
Bot: Missouri (État) et Indiana et Kentucky et Iowa et Wisconsin

Vous: Qui était l'épouse du président américain Lincoln ?
Bot: Mary Todd Lincoln

Send



Version graphique (gui activé)

En plus du système de questions / réponses dont fait l'objet ce projet, nous avons développé une interface graphique basique pour permettre une utilisation plus conviviale pour l'utilisateur. Nous nous sommes pour cela basés sur la bibliothèque standard de Python **tkinter**. L'interface devrait ainsi s'afficher de la même façon sous une machine pouvant faire tourner un interpréteur Python quel que soit le système d'exploitation. L'interface permet la saisie des questions, la consultation des réponses et l'activation des fonctionnalités vocales (plus de précisions sur ces fonctions dans la section suivante). L'initialisation de l'interface graphique est entièrement effectuée dans la fonction « **demarrage_gui** » (uniquement si l'option gui est activé, se reporter à la partie « *configuration du projet* » de ce rapport). L'affichage des questions ainsi que des réponses sur l'interface graphique est géré par la fonction « **send** ».

Fonctionnalités vocales

Reconnaissance vocale

La fonctionnalité de reconnaissance vocale que nous avons intégrée permet de comprendre une question posée à l'oral **en français** grâce à un microphone. Pour activer cette fonctionnalité, cliquer sur l'icône rouge représentant un microphone en bas à droite de l'interface graphique. **Merci de noter qu'il est absolument nécessaire d'avoir activé l'interface graphique (gui) pour activer l'ensemble des fonctionnalités vocales intégrées.** Afin d'améliorer la reconnaissance de la question, un message indiquera l'ajustement du niveau de bruit avant de vous inviter à poser la question. Un délai de 5 secondes sera alors laissé à l'utilisateur pour poser la question à l'oral. L'ensemble des opérations de reconnaissances vocales sont codées dans la fonction « **voix** ». La conversion de la question orale en français vers du texte s'effectue à l'aide des outils mis à disposition par *Google*.

Bot: Ajustement du niveau de bruit... veuillez patienter
Bot: Enregistrement en cours pour 5 secondes (posez votre question)
Bot: Fin de l'enregistrement
Bot: Reconnaissance du texte en cours...

Vous: qui est la reine du Royaume-Uni ?
Bot: Élisabeth II

*Note : Si vous n'apercevez aucune icône de microphone visible sur l'interface gui, merci de vérifier que les fonctionnalités vocales sont activées. Se reporter à la section « **Configuration du projet** » de ce rapport pour plus d'informations.*

Lecture vocale

Lorsqu'une question est posée à l'aide de la reconnaissance vocale, la réponse est apportée **à la fois à l'écrit et à l'oral** via les haut-parleurs de la machine. Les opérations concernant la lecture vocale sont codées dans la fonction « **parler** ». La conversion de la réponse écrite vers une réponse orale en français est également effectuée à l'aide des outils mis à disposition par *Google* (*Google Text-to-Speech*). La bibliothèque mise à disposition par *Google* ne permettant que la création d'un fichier *mp3* à partir du texte lu (et non la lecture sur haut-parleur), nous lisons donc ce fichier audio à l'aide d'un lecteur audio externe inclus dans le package **pyglet** avant de directement supprimer le fichier *mp3* du disque.

Evaluation de notre système

Le système conçu répond correctement à 100% des questions proposées dans le jeu de donnée « *question.xml* » (à quelques mises à jour près car les réponses aux questions ont évolué depuis l'écriture du jeu de donnée, et une anomalie dans la base DBpedia renvoyant une date erronée pour la question « Quand se déroula la bataille de Gettysburg ? » [cela fonctionne correctement si remplacement par « bataille de Verdun » par exemple]). Pour cette raison la précision, le rappel et la F-mesure pour chaque question est de 1.

Ce résultat est cependant à relativiser par le fait que nous avons conçu notre système de questions / réponses pour qu'il réponde spécifiquement à toutes les questions de ce jeu de donnée (mais pas exclusivement). En effet si le système devrait très bien s'en sortir sur les questions types mentionnés plus tôt dans ce rapport, il risque d'avoir des réponses beaucoup plus aléatoires ou ne pas donner de réponses du tout sur des questions qui sortent des patterns que nous avons conçus.

Améliorations possibles

Nous avons tenté de livrer le projet le plus abouti possible dans le temps qui nous était imparti mais il est bien sur toujours possible de suggérer des pistes d'améliorations du programme. La principale piste à explorer concernerait les expressions régulières afin d'extraire les bonnes données de la question. En effet, s'il est vrai que des bonnes réponses sont apportées sur la totalité des questions du jeu de données fourni via le fichier « *questions.xml* », le système conçu n'est pas forcément aussi performant sur des questions traitant complètement d'autres sujets ou même utilisant un vocabulaire différent. Il conviendrait alors d'ajouter de nouvelles expressions régulières afin de matcher de nouveaux types de questions mais aussi de compléter et de généraliser d'avantage celles déjà existantes. Nous n'avons en particulier pas exploité la totalité des entités proposés dans la base DBpedia et les nouvelles expressions régulières créées pourraient ainsi exploiter davantage de catégories référencées dans cette base.

Nos expressions régulières se basent d'ailleurs sur une stratégie de type « *exact match* » repérant ainsi des mots précis dans les questions de l'utilisateur, il pourrait être intéressant d'explorer d'autres solutions se basant par exemple la *distance de Levenshtein* ou encore sur la base *WordNet*. Nous avons rapidement expérimenté ce type de stratégie mais nous obtenions de manière générale des réponses plus pertinentes en matchant exactement les mots cherchés. Cela semble particulièrement vrai en français car la plupart des outils repérant les similarités entre les mots semblent, de manière générale, mieux fonctionner en anglais.

Enfin une dernière piste d'amélioration possible consisterait à utiliser une base de données plus à jour pour le renvoi de réponses (les réponses datant, pour la plupart, de quelques années) voir même de développer un outil permettant d'extraire en temps réel les informations disponibles sur le web. L'intégration de « *DBpedia Live* » au projet pourrait également être intéressante car cet outil permet de relever les modifications qui ont eu lieu sur Wikipédia depuis la dernière mise à jour de *DBpedia*.