EISD

meleeleolaoleolelele

Professeurs :
Thomas Lavergne
Sophie Rosset

Introduction

Dans le cadre de la matière Extraction d'Informations et Système de Dialogue (EISD), nous avons réalisé un projet qui consiste en la création d'un système de dialogue auquel on peut poser des question sur un thème choisi.

Le projet est décomposé en deux parties : la partie extraction de donnée dans laquelle on crée un programme qui lit des textes dans le but d'en retirer des informations non-structurées et de créer avec ça une base de donnée, et la partie système de dialogue qui doit comprendre les questions de l'utilisateur et lui répondre avec les informations de la base de donnée, tout en proposant une expérience qui soit la plus intuitive et immersive possible.

Concernant le thème, nous avons décidé de nous diriger vers le jeu Super Smash Bros. Melee, plus précisément la scène compétitive de celui-ci.
En effet, c'est un jeu que nous affectionnons tous et qui malgré les sorties de nouveaux Super Smash Bros. est toujours très joué compétitivement. De plus, un wiki existe recensant tous les joueurs de Melee ainsi que de multiples informations sur ceux-ci (nom, pseudo, nationalité, rang, tournois etc.) dont nous pouvions nous servir pour la partie extraction de données.

C'est donc grâce à ces différents éléments que nous avons créé le système de dialogue nommé *meleeleolaoleolelele*, un merveilleux chatbot spécialisé dans la scène compétitive de Melee. L'utilisateur peut lui poser plusieurs types de questions sur les différents joueurs pro, que l'on explicitera un peu plus tard dans ce rapport.

Table des matière

Introduction **2**

Table des matière **3**

Répartition des tâches **4**

Récupération des corpus de texte **5**

Extraction et stockage des données **6**

Système de dialogue **7**

Répartition des tâches

La répartition des tâches à été faite comme suit :

Récupération des corpus : Mathias, Nathan
Extraction des données : Nathan, Kévin
Système de dialogue : Mohamed, Mathias
Traitement des textes : Kévin

Récupération des corpus de texte

Nous avons choisi d'automatiser la récupération du corpus pour essayer d'avoir maximum de textes et d'information. La récupération des textes se fait exclusivement sur le site https://www.ssbwiki.com/ .

Pour cela, nous avons utilisé Python et Lynx. On commence par récupérer le texte de la page du wiki qui liste les meilleurs joueurs de l'année précédente avec puis notre script python va lire la page et se rendre dans tous les liens hypertextes pertinents qu'il trouve pour télécharger les textes des pages auxquels ils emmènent (donc les pages concernant les meilleurs joueurs). Le script peut continuer de manière récursive indéfiniment.

On s'assure que le script ne télécharge que des pages dont l'url contient les mots clés Smasher (indiquant un joueur) ou Tournament, et qu'il évite les fichiers type .jpg, .gif ou encore les pages php. Au final, nous avons des profils dans la base de donnée pour 152 joueurs sur 246 fichiers téléchargés.

Extraction et stockage des données

Pour l'extraction des données, nous voulions faire une base de donnée à deux niveaux, avec d'un côté, les informations sur les joueurs, et de l'autre les informations sur les tournois. Malheureusement, lors de l'extraction des données, nous avons eu que très peu d'informations concernant les tournois, ce qui ne nous permettait pas de remplir convenablement une base de donnée. Nous avons donc ensuite essayé d'utiliser une API développée par des fans sur internet mais sans succès.

En conséquence, nous avons fini par décider de nous focaliser sur l'extraction d'informations concernant les joueurs. La base de donnée qui est dédié à cette tâche contient actuellement 5 types d'informations : le pseudo, les mains, la localité, les surnoms et le classement.

Pour ce qui est de la partie non structurée, ces informations sont récupérés dans des textes non structurés récupérés en suivant les processus de récupération du corpus mentionné précédemment. La détection de chacune de ces informations se fait quasiment exclusivement par le biais de patterns, à l'exception de l'extraction des mains qui nécessite en plus de patterns dédiés un lexique des personnages jouables dans le jeu afin de pouvoir déterminer ce qui est un personnage du jeu jouable potentiellement main de joueur. Une fois toutes les informations récupérées, la base de données en résultant est transcrite dans un fichier .txt pour être chargée dans la partie dialogue. Le résultat d'une base de donnée en structuré est contenu dans le fichier DbbContextFirst.txt dans le dossier dark. Le premier script s'occupant de l'extraction non structuré étant parsing.lua dans le dossier dark.

En ce qui concerne les données structurés, elles permettent de compléter et vérifier les 5 types d'information précédents, mais permettent aussi de récupérer des informations comme l'argent gagné, la date de naissance ou encore les sponsors. Le résultat de la base de donnée finale est contenu dans le fichier DbbContextSecond.txt dans le dossier dark. Le deuxième script s'occupant de l'extraction structuré étant ComplementData.lua dans le dossier dark.

Pour ce qui est des statistiques concernant l'extraction des données, après extraction des données structuré, on se retrouve avec le profil de 152 joueurs avec un nombre de données de 446, ce qui est plutôt raisonnable. Après récupération des données structuré, la base de donnée se retrouve bien plus remplie avec 47565 données ajoutées dont 70 qui étaient des modifications de données déjà existantes, pas forcément fausses mais peut-être plus explicites. Il y a aussi 174 données qui ont été extraites en non structuré et validé en structuré. On peut donc voir que près d'un tier des informations extraites en non-structuré sont bonnes, pour le reste, l'information est généralement bonne d'après nos observations mais n'est pas vérifiable.

Problèmes et solutions

Même si notre corpus de joueur hors-ligne contient des pages pour environ 240 joueurs, certaines pages sont très incomplètes, notamment celles des joueurs les moins connus du wiki. Cela nous a malheureusement limité à choisir des types d'informations qui pourraient être mis en commun entre les pages et qui seraient la plupart du temps présentes sur lesdites pages. De cette constatation en a découlé le choix des informations décrites précédemment, et même s'il existe quand même des pages pour lesquelles ces informations sont en partie indisponibles, cela n'est pas réellement un problème vu le nombre de pages dans le corpus contenant ces dernières.

De plus, même si la base de données contient quand même un bon nombre de joueurs, il n'a malheureusement pas été possible de prendre en compte tous les joueurs de notre corpus à cause de la nature irrégulière des pseudos des joueurs. En effet, les pseudos des joueurs peuvent posséder n'importe quel type de caractères, y compris des nombres, des caractères spéciaux, des noms composés (comme Bizarro Flame), des noms avec diminutifs (Dr. par ex), etc. En conséquence, les pseudos ne peuvent pas être trouvés directement mais doivent plutôt être déduits selon la structure des phrases contenant ces mêmes pseudos, et une solution possible aurait pu être de considérer la structure grammaticale de ces phrases en plus des formulations revenant souvent et des mots clés caractéristiques de l'introduction d'un joueur.

Système de dialogue

Description et fonctionnalités

Le système de dialogue que nous avons développé répond après chaque phrase entrée par l'utilisateur. Il va tenter de détecter une question dans l'entrée de l'utilisateur en utilisant des paternes et, si besoin, d'autres informations clés dont il aura besoin pour répondre comme par exemple le nom d'un joueur ou d'un personnage du jeu. Ces dernières sont détectées au moyen de lexiques. L'utilisation de paternes permet de garder une certaine souplesse pour la détection et permettre de nombreuses formulations pour une même question.

Il est fait en sorte que les ces mots clés soient détectés en laissant une certaine marge d'erreur à l'utilisateur, il faut par exemple que le nom d'un joueur soit reconnu même s'il a fait une petite faute de frappe. Pour cela, on regarde la distance de Levenshtein entre les mots entrés avec les mots des lexiques, ce qui permet de trouver un mot clé qui aurait été écorché. On charge aussi les lexiques en minuscules en plus de la casse normale pour qu'ils fonctionnent si l'utilisateur ne met pas de majuscules.

Le système implémente aussi un historique pour garder en mémoire les questions qui ont été posées et les réponses qui ont été données. Cela permet par la suite à l'utilisateur de poser des questions sans spécifier explicitement le sujet ou le type de question. Par exemple si l'utilisateur a posé une question sur Mango (ex: *Who is Mango ?*), il pourra ensuite poser la question *What about Armada ?*, ce qui donnera le même résultat que s'il avait demandé *Who is Armada ?.*

Notre système essaie le plus possible d'avoir un comportement "humain" pour augmenter l'immersion de l'utilisateur. Par exemple, il relancera l'utilisateur après chaque réponse pour l'inciter à poser une nouvelle question. Il signale lorsqu'il ignore la réponse à une question, qu'il ne connaît pas un joueur ou qu'il n'a pas compris la question qui a été posée. Enfin il réagit à certaines situations, par exemple, il s'en ira si on lui dit au revoir ou si on lui fait des avances déplacées.

Voici une liste de questions que le système reconnaît et que vous pouvez vous amuser à essayer :

What is Leffen's main
Who is Leffen's main ?
What characters does Leffen play ?
Which characters does Leffen play ?
Who plays Falco ?
Who is Leffen ?

Who's PewPewU ?
Where is Armada from ?
Where does Zain come from ?
What is Leffen's nationality ?
What is Mango's country ?
Where does Armada live ?
What is Mango's nickname ?
what's mango ranked ?
what is mango's rank ?
what is mango often called ?
who plays fox ?

who is leffen's main ?
what about Armada ?
Who is the best between them ?
where is he from ?

Who is the best player ?
When is Leffen's bday ?
When was Mango born ?
What is Armada's team ?
What's Armada's sponsor ?
Who is the best between armada and mango ?
And leflen ? (faute de frappe voulue)

Bye !

Problèmes et solutions

Pour levenshtein il a fallu trouver un juste milieu à la marge d'erreur qu'on laisse à l'utilisateur. Si on autorise une distance levenshtein trop petite, les similarités entre l'erreur et le mot clé ne sont jamais détectées, mais si au contraire on autorise une distance trop grande, le système peut confondre un mot correct qui fait partie de la question pour un mot clé. Par exemple on peut se retrouver avec l'interaction suivante:

user : What **are** his mains ?
system : Did you mean **Axe** ?

Nous avons décidé pour empêcher cela d'ajouter des mots exceptions que le système n'essaiera pas de corriger (comme *are* par *Axe* ou *main* par *Zain*).

Parfois des types questions peuvent se ressembler ou être inclus les uns dans les autres. Par exemple "*Who is Leffen*" est inclu dans "*Who is Leffen's main ?*", le système détectera alors aussi la première dans le cas où l'utilisateur pose la seconde. Il faut donc

faire attention lors de la détection à tester les différents types dans le bon ordre pour être sur que l'un ne soit pas masqué à chaque fois par un autre.

Ici, il faut d'abord tester la présence de *Who is Leffen's main* puis celle de *Who is Leffen*.

Pour répondre aux questions il faut s'assurer que la donnée est présente dans la BD car sinon on risque de faire crasher le système en essayant d'afficher nil. Nous avons essayé pour palier ce problème de gérer les cas où la donnée est nil dans certaines questions.