DS: Piton ChatBot

Enrico Mosca, Alberto Lillo, Alessandro Saracco

July 2022

1 Introduzione

DS per apprendisti Streghe o Stregoni.

Il progetto riguarda la realizzazione di un Dialog Speech tra Severus Piton (il personaggio della saga Harry Potter) che ricopre il ruolo di Professore ed interagisce con l'utente, quest'ultimo nel ruolo di interrogato.

Durante l'esecuzione del programma all'utente verranno chiesti gli ingredienti di tre pozioni e successivamente verrà assegnato un voto complessivo dell'interrogazione.

Il Dialog Speech è stato trattato interamente in lingua inglese.

L'iniziativa è del sistema che deve interrogare, chiedendo all'utente il proprio nome e la casata di appartenenza, successivamente inizia l'interrogazione in cui all'utente viene chiesto di indicare gli ingredienti di una pozione specifica.
Il DM quindi, deve interrogare sugli ingredienti ancora mancanti, eventualmente proponendo risposte vere o false.

2 Analisi

Il linguaggio utilizzato per la realizzazione del programma è il linguaggio **Python**.

Per la realizzazione di questo progetto si è scelto di utilizzare un **approccio con dipendenze** utilizzando quindi il parser a dipendenza **Spacy** che consente di parsificare e tokenizzare frasi al fine di modellarne la gestione.

Per quanto riguarda la generazione del **Text-Plan** ed il **Sentence-Plan** è stata utilizzata la libreria **Simple-NLG**[1] in lingua inglese.

Infine, per la gestione delle pozioni e relativi ingredienti, è stato scelto di appoggiarsi ad un file JSON, struttura che permette una flessibile manipolazione dei dati.

[1] `https://github.com/simplenlg/simplenlg`

2.1 SpaCy

SpaCy[2] è una libreria open-source per l'elaborazione avanzata del linguaggio naturale.
Presenta modelli di rete neurale convoluzionale per la codifica di parti del discorso, l'analisi delle dipendenze , la categorizzazione del testo e riconoscimento di entità nominative (NER).

SpaCy permette di poter scegliere il grado di accuratezza relativo alla conoscenza di base che la libreria può utilizzare, pertanto si è scelto di adottare il modello più accurato a disposizione, al fine di ottenere un'accuratezza maggiore durante le fasi di parsificazione delle frasi.

```
spacy.load("download en_core_web_trf")
```

Come detto prima questa libreria è stata utilizzata per la parsificazione e tokenizzazione di una frase, infatti è stata implementata la componente relativa al POS Tagging, di seguito un estratto del codice relativo:

```
def get_ingredient(frase, aiuto):
    if aiuto:
        return
    sent_dict = {}
    frase_parsificata = nlp(frase)
    position = 'nsubj'
    for chunk in frase_parsificata.noun_chunks:

        if 'ingredient' in chunk.text and chunk.root.dep_ == 'nsubj':
            position = 'attr'
        if chunk.root.dep_ == 'nsubj' or chunk.root.dep_ == 'attr':
            sent_dict[chunk.root.dep_] = chunk
    displayParser(frase_parsificata)
    return sent_dict[position]
```

In questo metodo viene data in input una frase e successivamente parsificata:

```
frase_parsificata = nlp(frase)
```

A questo punto, andando a ciclare sui token creati si ha la possibilità di poter osservare quali sono le componenti logiche della frase, infatti si va a controllare dove si trova la parola **ingrediente**, ovvero se è soggetto (nsubj) oppure complemento (attr).

Di seguito un esempio visivo (realizzato con il modulo **displacy**) della parsificazione relativa alle frasi:

- ***An ingredient is Ashwinder's egg***: attr
- ***Ashwinder's egg is an ingredient***: nsubj

[2] https://spacy.io/

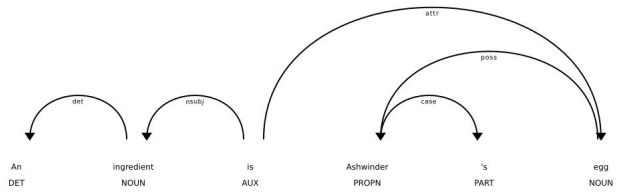

Figure 1: Risultato della parsificazione nel caso in cui la parola ingrediente sia il soggetto della frase

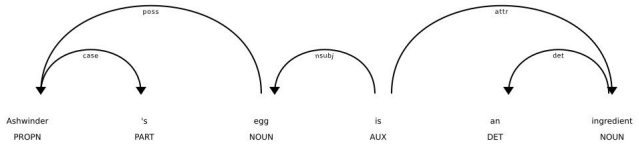

Figure 2: Risultato della parsificazione nel caso in cui la parola ingrediente sia l'attributo della frase

Questa libreria è stata utilizzata anche per fare **Named Entity Recognition**, infatti quando il DS chiederà di inserire il nome dell'utente, l'input inserito verrà passato alla funzione ***parser_ne*** di seguito riportata:

```
def parser_ne(frase):
    frase_parsata=nlp(frase)
    dict = {}
    for token in frase_parsata.ents:
        dict[token.text]=[token.text, token.start_char,
        token.end_char, token.label_]
    for key, value in dict.items():
        if("PERSON" in value):
            return key
    return None
```

Tale funzione andrà, come al solito, a parsificare la frase dove verranno individuate le entità di quest'ultima.

In particolare abbiamo controllato che nella frase ci fosse l'entità **PERSON** per poter distinguere i casi in cui l'utente inserisca correttamente un nome proprio di persona, altrimenti verrà avviato un ciclo *while* che imporrà all'utente di inserire una frase con all'interno un nome proprio di persona.

3 Dialog-Manager

3.1 Frame-Based

Il chatbot è a conoscenza delle pozioni e degli ingredienti che le compongono, grazie a una struttura JSON contenuta all'interno del file ***pozioni.json***.

Ogni pozione infatti è caratterizzata da tre parametri: nome, difficoltà e la lista degli ingredienti. Di seguito un esempio della struttura

```
"Memory potion":[
        2,
        ["Jobberknoll's feathers",
         "powder sage",
         "snowdrop"]
        ]
```

Inoltre esistono altre due liste, per gestire il discorso:

- *pozioni_chieste*: viene riempita con la pozione chiesta
- *ingredienti_indovinati*: viene riempita quando un ingrediente è stato trovato

Il primo frame viene riempito per chiedere tutte le pozioni, mentre il secondo serve per chiedere all'utente tutti gli ingredienti che compongono la pozione ed evitare che vengano ripetuti.

3.2 Backup-Strategy

Come backup-stategy è stata adottata la scrittura su file. Ogni domanda viene scritta all'interno di una lista sul file ***questions.txt***. Questo ci permette quindi di rifare la domanda all'utente ogni qualvolta questo scrive in input una frase contente le parole "repeat", "again" o "what?".

Es: "Can you repeat the question?"

A questo punto il chatbot caricherà la lista dal file e andrà a leggerne l'ultima domanda, riproponendola.

3.3 Memory

Per quanto riguarda la memoria il chatbot salva, ad ogni risposta dell'utente, l'input dato all'interno di un file di log chiamato ***answers.txt***.

Inoltre immagazzina all'interno della lista ***ingredienti_indovinati*** gli ingredienti già indovinati. Questo per far si che l'utente non risponda sempre con lo stesso ingrediente. Nel caso in cui vengo riproposto lo stesso ingrediente, vengono detratti 20 punti e la domanda non è valida.

3.4 Ciclo delle domande

Il ciclo delle domande può essere scomposto in due macro cicli principali:

- Ciclo sulle pozioni

- Ciclo sugli ingredienti della pozione scelta

Di seguito una sua schematizzazione che andremo ad analizzare

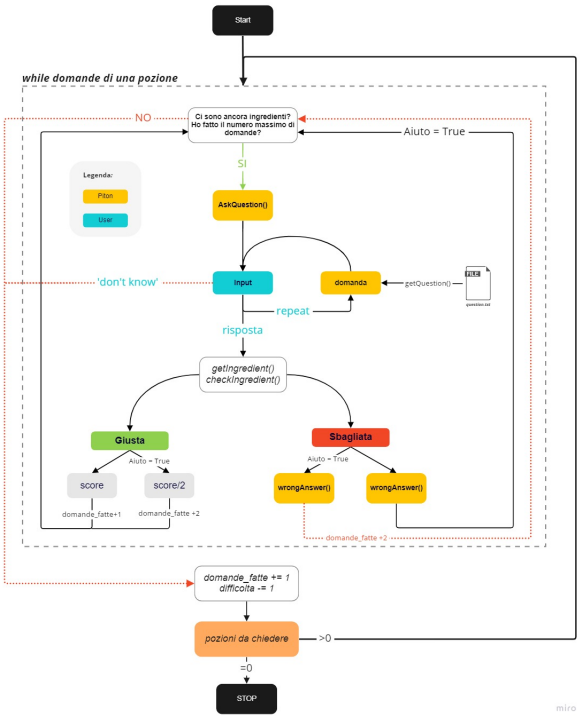

Figure 3: Schema del ciclo delle domande

Il bot prima di tutto chiederà tutte e 3 le pozioni e farà tante domande quante sono gli ingredienti di ogni pozione più uno. Abbiamo pensato di permettere all'utente di sbagliare una volta per pozione. Ogni volta che viene chiesta un ingrediente l'utente può rispondere con:

- La risposta con ingrediente
- Rispondere con "I don't know"
- Può chiedere di ripetere

Risposta con ingrediente

Nel caso della risposta corretta il chatbot assegnerà i punti e passerà alla pozione successiva.
In caso di risposta sbagliata, verrà proposto una domanda trabocchetto (vero o falso).
Il sistema infatti è a conoscenza di tutti gli ingredienti che compongono tutte le pozioni, tramite il metodo ***get_all_ingredients()***. Ottenuta la lista, sceglierà un ingrediente random, il quale viene passato a Simple-NLG che genererà la frase corrispondente.
A questo punto l'utente potrà rispondere solo con yes/no, se la risposta è corretta vengono assegnati metà dei punti. L'utente può comunque rispondere con "don't know".

"I don't know"

Nel caso in cui l'utente non sappia rispondere, il chatbot passerà direttamente alla pozione successiva, assegnando uno score negativo

Ripeti

In questo caso Piton riproporrà l'ultima domanda fatta

4 Generazione

4.1 Simple-NLG

Per quanto riguarda l'implementazione di Simple-NLG, prendo in esame due delle casistiche create.

4.1.1 printAskPotion

Il metodo ***printAskPotion*** genera le domande poste da Piton per quanto riguarda le pozioni e ingredienti.
Tutte le frasi generate partono da una frase base che viene creata all'inizio del metodo, ed è "What is the ingredient?"
I passi per creare questa frase sono:

- Creiamo due **NounPhrase**, la prima con l'articolo ***the*** e l'oggetto ***potion***, la seconda con l'articolo ***the*** e l'oggetto ***ingredient***.
- Per formulare la frase, creiamo una **Clause** contenente come soggetto la **NounPhrase** di ***ingredient*** e il verbo ***be***.
- Settiamo la **Feature** per rendere la frase interrogativa usando il metodo ***setFeature()*** in cui andiamo ad impostare
  - ***featureName*** con **INTERROGATIVE_TYPE**
  - ***featureValue*** con **WHAT_OBJECT**

Al momento abbiamo settato la frase base.
Da qui in poi questa frase muterà a seconda del punto in cui si trova all'interno della conversazione.
In particolare la frase verrà modificata nei seguenti modi a seconda del caso:

- "Let's start with the ***potion***, what are the ingredients?": viene fatta all'inizio dell'interrogazione alla prima pozione
- "What are the ingredients of the ***potion***?": viene fatta dalle domande successive
- "What is the last ingredient?": viene fatta quando manca un solo ingrediente

Con ***potion*** variabile che cambia a seconda della pozione richiesta da Piton.

4.1.2 printAskIngredients

Il metodo ***printAskIngredients*** genera le domande che Piton farà nel mentre che lo studente elenca gli ingredienti della pozione.
Il metodo prevede che Piton tenga conto degli ingredienti detti dallo studente e ogni volta lo aggiorni di quanti ne mancano man mano che si va avanti, chiedendo ogni volta quali sono i rimanenti. La frase verrà messa al plurale quando manca al più un ingrediente.

"#ingredienti ingredients are missing."

5 Valutazione

Analizziamo ora dei dialoghi. (nota: >> identifica l'input dell'utente)

```
Good morning.
I am Severus Piton.
What is your name?
    >>my name is Enrico
What is your house?
    >>Gryffindor
Ok let's proceed!
Enrico, did you study?
    >>yes
```

Questo dialogo avviene all'inizio di ogni interrogazione. L'interrogazione non inizia affinché l'utente non specifica il nome e la casata.
Successivamente viene chiesta la prima pozione.

```
Let's start with the Fire breathing potion,
what are the ingredients?
    >>fire seeds is an ingredient
```

L'utente a questo punto può rispondere con l'ingrediente oppure con 'non lo so'

```
4 ingredients are missings.
What are the other ingredients?
    >>I don't know

What are the ingredients of Memory potion?
```

In questa interazione notiamo che il sistema ci chiede gli ingredienti mancanti in quanto ne è stato indovinato già uno, ma rispondendo 'I don't know', proseguirà con la prossima pozione.
Qualora l'utente chiederà di ripetere, verrà riproposta la domanda.

```
What are the ingredients of Memory potion?
    >>Can you repeat?
What are the ingredients of Memory potion?
```

Infine quando sono stati trovati quasi tutti gli ingredienti, tranne uno. Il sistema proporrà la seguente domanda:

```
What is the last ingredients?
    >>The last ingredient is Jobberknoll's feathers
```

Terminate tutte le pozioni, concluderà assegnando lo score alla relativa casata

```
You could have done better.
I award 23 points to Gryffindor.
```

Il sistema presenta delle limitazioni. Prendendo in considerazione diversi tipi di risposta alla domanda "What's your name", è emerso, ad esempio, che non è in grado di riconoscere i nomi in italiano, obbligandoci a forzare il chatbot eludendo il NER Tag. Qui sotto vediamo infatti, che pur essendo una frase corretta, Paolo non viene riconosciuto dal NER come "Person", chiedendo nuovamente di dire il proprio nome.

```
What is your name?
    >>my name is Paolo
You must tell me your name.
```

Dall'analisi di alcuni dialoghi, abbiamo estrapolato quattro tipi di risposte possibili durante l'interrogazione sugli ingredienti. L'utente infatti può rispondere con 'non lo so' oppure dando una risposta:

- con solo l'ingrediente
- con l'ingrediente come soggetto
- con l'ingrediente come complemento

Il sistema dunque è limitato a questa struttura di risposta. Non sarà dunque in grado di comprendere risposte che usano ad esempio un sinonimo di 'Ingrediente'. Inoltre è possibile scrivere solo un ingrediente alla volta.
All'inserimento dell'input non viene controllato il contesto, ma qualora l'utente scriva qualcosa di inatteso, risponderà con un commento che obbliga l'utente a rispondere alla domanda. (Vedi caso di errore nel nome, o nome non inglese).
Il DS gestisce la non risposta durante l'interrogazione come un 'non lo so', mentre per quanto riguarda le domande iniziali, propone una frase per informare l'utente di inserire un nome valido.

```
What is you name?
    >>
You must tell me your name.
    >>Enrico
What is your house?
    >>
You must tell me a valid house name!
    >>Gryffindor
Ok let's proceed!

Enrico, did you study?
    >>
Don't make fun of me! Tell me if you have studied!
    >>Yes
....
Well, now we will find out ..
```

Avendo utilizzato Spacy, l'utente è libero di scrivere qualsiasi tipo di risposta, l'importante è che contenga almeno la parola "ingrediente".

6 Trindi Ticklist

Di seguito commentiamo brevemente alcuni punti della Trindi Ticklist.

- **Q1: L'interpretazione degli enunciati è sensibile al contesto?**
  I risultati vengono in interpretati nel dominio di utilizzo, per cui il sistema lavora solo nel contesto dell'interrogazione..

- **Q2: Il sistema è in grado di gestire le risposte alle domande che forniscono più informazioni di quelle richieste?**
  Si, è in grado di recuperare l'ingrediente della pozione da una frase che contiene altre parole oltre al nome dell'ingrediente.

- **Q3: Il sistema è in grado di gestire le risposte alle domande che danno informazioni diverse da quelle effettivamente richieste?**
  Si, è in grado di gestire risposte fuori contesto o sbagliate, rispondendo con frasi che indicano all'utente di aver sbagliato, per esempio: *"That's a pity, try again"*

- **Q4: Il sistema è in grado di gestire le risposte alle domande che forniscono meno meno informazioni di quelle effettivamente richieste?**
  No, non è previsto che l'utente possa scrivere parzialmente il nome di un ingrediente, pertanto non è attualmente presente un sistema di riconoscimento parziale dell'input

- **Q5: Il sistema è in grado di gestire informazioni specificate negativamente?**
  No, in quanto il sistema impersonifica un professore che interroga, per tanto non è consueto che l'utente risponda con una frase negata, ma solo con affermazioni positive

- **Q6: Il sistema è in grado di gestire una domanda senza risposta?**
  Si, gestisce una risposta "vuota" come gestirebbe una risposta sbagliata o fuori contesto, quindi viene trattata come una risposta errata (come in una vera interrogazione) per cui verranno stampate delle frasi come nell'esempio della domanda **Q3** , fin tanto che 'utente non risponde correttamente o con una risposta contestuale (es. *"I don't know"*)

- **Q7: Il sistema è in grado di gestire informazioni incoerenti?**
  Si, rispondendo come indicato nella domanda **Q3**

- **Q8: È possibile ottenere un tutorial sul sistema per quanto riguarda il tipo di informazioni che il sistema può fornire e quali sono i vincoli di input?**
  Durante l'esecuzione del sistema no, ma nella relazione sono indicati tutti i dettagli ed eventuali vincoli di risposta che l'utente dovrebbe rispettare

- **Q9: Il sistema pone solo domande di follow-up appropriate?**
  Si, il sistema fa delle domande in base alle risposte dell'utente, per cui segue il flusso di input dall'utente

- **Q10: Il sistema è in grado di gestire input rumorosi?**
  No, il sistema comprende soltanto frasi di senso compiuto

7 Conclusioni

In conclusione, il progetto ci ha fatto capire quanto sia difficile analizzare e comprendere il linguaggio.
In particolare, abbiamo notato quanto sia complicato gestire tutte le flessioni linguistiche della frase e tutte le sue possibili combinazioni. Infatti ogni volta che veniva scritto un metodo che teneva conto di una regolarità, lo stesso non teneva conto di un'altra possibile combinazione.