

Piton Dialogue System in Python3



Sciandra Lorenzo, Torrielli Federico, Staffieri Michele¹

¹Dipartimento di Informatica

2456 Words

Student ID: 859704, 858839, 869241

Keywords: Dialogue, NLP, python

19th May, 2022

Abstract

Implementazione del secondo esercizio proposto dal Professor Mazzei per la prima parte del corso magistrale Tecnologie del Linguaggio Naturale. In questa documentazione verranno presentate le nostre scelte implementative per la realizzazione di un dialogue system frame based in grado di fare sia analisi che generazione del linguaggio naturale, impersonando il personaggio d'invenzione della Rowling.

1 Introduzione

1.1 Motivazioni

Una delle parti più interessanti quando si parla di *Natural Language Processing* sono sicuramente i sistemi di dialogo, dato una loro buona implementazione prevede ottime conoscenze sia di analisi che di generazione del linguaggio naturale. In questo progetto abbiamo voluto sperimentare utilizzando sia ciò che abbiamo imparato dal corso sia le risorse ritenute da noi interessanti.

Nonostante il problema fosse tutt'altro che facile e realizzabile da moltissimi lati, si è voluta dare un'impronta innovativa aggiungendo una sorta di personalità e rendendo quanto più umano il personaggio di Piton.

1.2 Installazione ed utilizzo

Il progetto è disponibile a [questo link](#). Qui in seguito, le istruzioni per installarlo e testarlo su una macchina basata su sistema Unix (Linux e MacOS like).

Da terminale, dopo aver scaricato ed estratto il progetto, eseguire i seguenti comandi nella cartella:

```
1 python3 -m pip install -r requirements.txt
2 spacy download en_core_web_md
3 python3 dialogue_manager.py
```

Nel caso in cui il comando python3 non esistesse, cambiare tutte le occorrenze di python3 con python (senza il 3).

1.3 Struttura del documento

In questa subsezione andremo a trattare della struttura del progetto. Per comodità, abbiamo diviso le sezioni nella maniera seguente:

- Introduzione: sguardo generale al progetto, frame, scelte implementative particolari, knowledge base;
- Analisi del linguaggio;
- Dialogue Manager;
- Generazione del linguaggio;
- Conclusione e risultati.

1.4 Articolazione del discorso

L'assistente in questione è Piton, ovvero il professore di Pozioni alla scuola di magia di Hogwarts. Si è all'interrogazione dell'esame finale del corso: per comodità gli argomenti sono stati ristretti alla conoscenza di tre pozioni. Il flow del discorso procede nel modo seguente:

- Il professore ha l'iniziativa: può fare domande, può spronare a continuare
- L'unico scopo dello studente è rispondere correttamente alle domande, ed andare avanti quanto più autonomamente possibile
- Il professore da, alla fine dell'interrogazione, una votazione ed un commento informativo

Viene rispettato, per quanto possibile, la personalità dell'assistente. Piton è un personaggio (apparentemente) egocentrico: non è particolarmente d'aiuto allo studente, se non in rari casi. Nelle sezioni successive verrà mostrato come è stato possibile codificare la sua personalità.

1.5 Struttura del codice

Il codice, per come è stato strutturato, viene diviso più o meno nelle sezioni appena trattate. Le classi di maggiore importanza sono `analysys.py`, `dialogue_manager.py`, `language_generator.py`, le altre classi hanno funzione di utility e boilerplate code per il resto del progetto. Le loro funzioni sono auto-esplicative.

Nel file `knowledge_base.py` è presente la base di conoscenza del nostro progetto:

- Le pozioni da noi scelte: pozione polisucco, invisibilità e della dimenticanza (Polyjuice, Invisibility, Forgetfulness)
- Una lista di parole utili utilizzate dal programma per analizzare le frasi da considerare in un discorso

- Liste di frasi da utilizzare randomicamente a seconda del mood di Piton

Il programma è **Frame-based**, ovvero man mano che si va avanti nel discorso, il frame si riempie di informazioni utili (o meno) che il programma utilizza per generare frasi diverse o per cambiare la sua personalità. Il Frame contiene di base due informazioni **la pozione di riferimento** (ovvero l'oggetto della domanda di Piton) e **la pozione in costruzione**. La struttura della pozione è molto semplice, essa contiene solo il suo nome e gli ingredienti che la compongono. Il frame contiene altre informazioni, oltre alle pozioni stesse, in particolare:

- `error_ingredients`: lista degli ingredienti che l'utente ha detto non appartenere alla pozione, ma invece appartengono;
- `external_ingredients`: lista degli ingredienti che secondo l'utente appartengono alla pozione, ma invece appartengono ad altre;
- `number_of_operations_made`: numero che indica quante iterazioni sul frame ci sono state;
- `wrong_number`: numero degli ingredienti che l'utente ha sbagliato in numero, ossia sono ingredienti che effettivamente appartengono alla pozione, ma sono stati detti con numero diverso.

2 Analisi del linguaggio

2.1 Intro

La classe `PhraseAnalysis.py` è incaricata dell'analisi delle frasi. Presa in input una frase, il metodo `correct_phrase` grazie ad una fuzzy correction rimpiazza e corregge le parole che presentano eventuali errori di battitura, la cosa ci interessa particolarmente poichè non vogliamo perdere parole chiave che saranno utili in seguito. A partire dalla frase corretta utilizziamo `spacy` che, presa in input l'intera frase restituisce l'albero delle dipendenze tra le parole, quindi tutti i tag del part-of-speech e delle dipendenze semantiche, oltre ad altre informazioni utili come la polarità e la presenza di interrogazioni. Sulla base di questo albero analizzeremo vari aspetti delle frasi. L'analisi viene effettuata su ogni frase data in input eseguito dall'utente, in questo caso lo studente interrogato da Piton.

2.2 Come sfruttiamo l'albero delle dipendenze

Come si vede dall'immagine [1](#), `spacy` costruisce l'albero delle relazioni sintattiche, ogni arco è etichettato con il tipo di relazione che vi è tra le parole. A partire dall'albero ottenuto con `spacy` andiamo a ricercare quattro caratteristiche della frase:

- Se è una domanda;

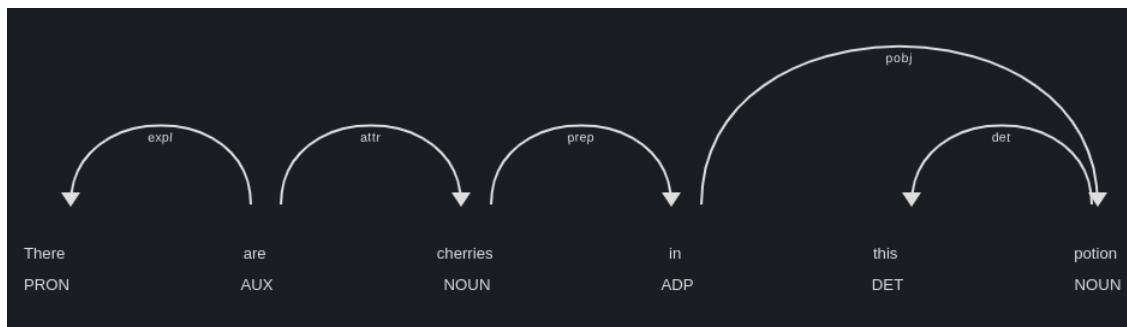


Fig. 1. Dependency parsing con Spacy

- Se è utile: cioè se la frase contiene ingredienti (in caso di utilità/non utilità verrà processata diversamente come si vedrà in seguito);
- Se contiene le parole yes/no;
- La sua polarità, cioè se è una frase affermativa o negativa;

2.3 Frasi interrogative

Per verificare se la frase sia o meno una domanda usiamo il metodo `check_if_question` in cui semplicemente verifichiamo se è presente un punto interrogativo o se la frase inizia con specifiche parole come "is", "are", "does", "do", "how" ecc...

2.4 Frasi utili

La verifica dell'utilità di una frase avviene nel metodo `check_if_usefull`, qui percorriamo tutta la frase e se troviamo parole utili marchiamo la frase come `useful` e aggiungiamo le parole utili ad una lista. La lista delle parole utili che cerchiamo all'interno di una frase è contenuta nella `knowledge_base` e contiene gli ingredienti. Se un ingrediente è formato da più parole come "valerian spring", "mistletoe berry" o "standard ingredient" analizziamo l'albero delle dipendenze per riconoscerle, per gli ingredienti nelle pozioni analizzate abbiamo riconosciuto tre tipi diversi di relazioni tra le parole:

- **amod**: modificatore aggettivale per ingredienti del tipo "mistletoe berry";
- **poss**: relazioni possessive per ingredienti come "bicorn's horn";
- **compound**: nomi composti per catturare ingredienti come "lether river water".

2.5 Frasi yes/no e polarità

Per alcune tipologie di domande che Piton porrà allo studente sarà richiesta una risposta di tipo yes/no per riconoscere queste frasi il metodo `check_yesno` verifica se la frase inizia con yes o con no o se non contiene affatto una di queste parole, si vedrà in seguito come reagirà Piton. Per la polarità invece analizziamo l'albero delle dipendenze, riconosciamo polarità negativa se:

- i) l'etichetta morph indica "Polarity=Neg";
- ii) l'etichetta dep indica "neg";
- iii) c'è un "no" che precede gli ingredienti per frasi del tipo "*There are no cherries in this potion*".

Diversamente la polarità sarà positiva

2.6 Esempio di analisi di una frase

Si mostra ora un esempio di analisi di una frase.

```
1  strin = PhraseAnalysis("There are cherries in this potion")
2  pprint(strin.phrase)
3  pprint(strin.dependency_tree())
4  print(f"Is useful: {strin.check_if_useful()}")
5  print(f"Polarity: {strin.check_polarity()}")
6  print(f"Is question: {strin.check_if_question()}")
```

In questo esempio è stato volutamente commesso un errore di battitura sull'ingrediente cherries per dimostrare il funzionamento della correzione automatica dei refusi citata all'inizio.

Output:

```
1  'there are cherries in this potion '
2
3  {'are': ('ROOT', 'are'),
4   'cherries': ('attr', 'are'),
5   'in': ('prep', 'cherries'),
6   'potion': ('pobj', 'in'),
7   'there': ('expl', 'are'),
8   'this': ('det', 'potion')}
9
10 Is useful: True
11 Polarity: True
12 Is question: False
```

Come ci aspettiamo l'analizzatore vede e corregge l'errore, riconosce la presenza dell'ingrediente, che non si tratta di una domanda è che la polarità è positiva.

3 Dialogue Manager

3.1 State Automa

Il professor Piton è stato da noi realizzato come una sorta di automa a stati finiti non deterministico con coda, con i seguenti stati:

- i) **Intro**: stato iniziale transitorio necessario solamente per far partire il dialogo con lo studente interrogato;
- ii) **Not Useful**: stato ricorrente in cui il professore si accorge che la risposta data dall'utente non è per lui utile. Questo accade quando non vi è riferimento esplicito a pozioni e/o ingredienti oppure quando l'utente propone una domanda. In questo scenario viene generata una frase *filler* con lo scopo di far procedere la conversazione;
- iii) **Fill The Frame**: stato ricorrente, complementare al precedente, in cui il professore si accorge che la risposta data dallo studente è utile e va a riempire il frame che sta mantenendo per poi generare la prossima domanda;
- iv) **Trabocchetto**: stato ricorrente in cui il professore fa un trabocchetto allo studente suggerendogli un ingrediente di un'altra pozione rispetto alla corrente selezionata per l'interrogazione;
- v) **Hint**: stato ricorrente in cui il professore fa un suggerimento allo studente chiedendogli se un argomento della pozione sia o meno presente;
- vi) **Questions**: stato ricorrente in cui il professore fa domande standard riguardo alla pozione scelta.

Per una maggiore comprensione verrà di seguito proposto uno schema di transizione degli stati [2](#).

3.2 Logica Implementativa

Come già accennato Piton è frame based nel senso che conduce la conversazione, mediante opportune domande, verso informazioni che ritiene necessarie per completare il frame dedicato alla pozione. Nello specifico le domande per essere utili sono rigenerate continuamente fino a quando non si verifica una delle seguenti condizioni:

- abbiamo una domanda generica sulla pozione e siamo nello stato *questions*;
- abbiamo una domanda che contiene un ingrediente di un'altra delle pozioni che ha in memoria e siamo nello stato *trabocchetto*;
- abbiamo una domanda che contiene un ingrediente della corrente pozione e siamo nello stato *hint*

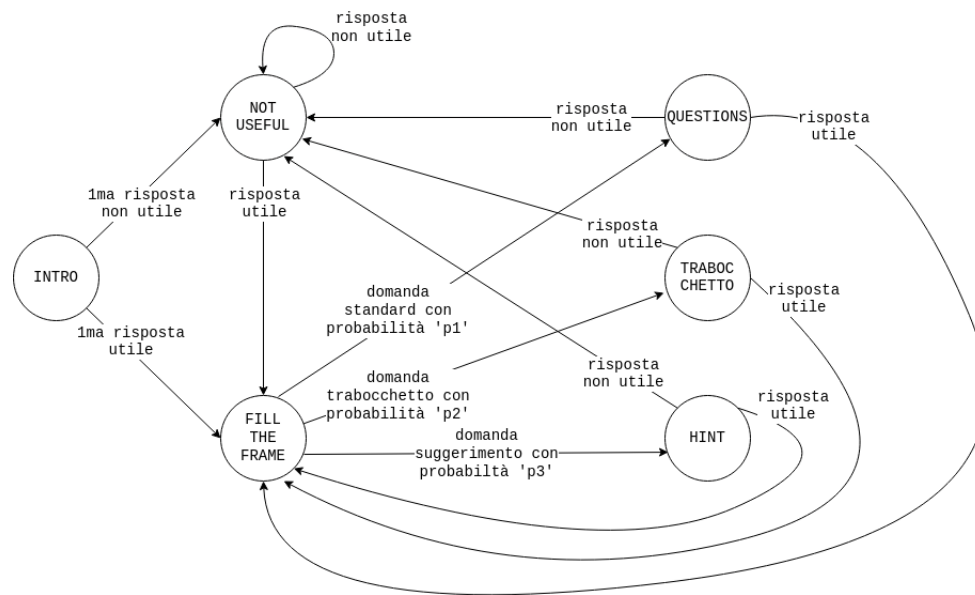


Fig. 2. Automa di Piton, il non determinismo risiede nella generazione delle domande che sono regolate da distribuzioni di probabilità

3.3 Mental State

Piton è dotato di uno **stato mentale** scelto tra **angry**, **neutral** e **happy** in maniera probabilistica all'inizio dell'esecuzione. Conoscendo il personaggio cinematografico queste probabilità sono a favore di un cattivo umore, che può però cambiare durante l'interazione se l'utente si dimostra molto ben preparato. Lo stato mentale impatta nell'esecuzione sia nella generazione, facendo affidamento a corpus di generazione completamente diversi e specifici per ogni stato, sia nella valutazione finale. Ad ogni turno di dialogo lo stato mentale può mutare sia in meglio che in peggio tenendo conto degli errori compiuti dall'utente, delle informazioni corrette restituite e della lunghezza della conversazione.

3.4 Terminazione

Il dialogo termina se si presenta una delle seguenti condizioni:

1. La conversazione procede per un numero di turni che è 3 volte il numero di ingredienti della pozione;
2. Il frame viene completato e l'utente ha detto correttamente o no tutti gli ingredienti della pozione;
3. L'utente specifica esplicitamente di non saper nulla.

3.5 Valutazione

La valutazione prende in conto tutto quello che è stato detto e genera il voto v con la seguente formula:

$$v = 31 - \alpha \cdot p$$

31 così parte con la lode, α è il moltiplicatore della penalità \mathbf{p} in base al suo stato mentale: $\alpha = 1.5$ se neutrale, $\alpha = 2$ se arrabbiato $\alpha = 1$ se felice.

$$\mathbf{p} = \#err + \frac{\#ext + \#plur}{2} + (3 \cdot t - 3)$$

con $\#err$ = numero degli errori, $\#ext$ = numero degli ingredienti esterni, $\#plur$ = numero dei singolari al posto dei plurali o viceversa e:

$$t = \frac{\#tentativi fatti}{\#tentativi perfetti}$$

Usiamo la funzione $y = 3t - 3$ (3) per trattare il precedente rapporto perchè ragionevole e facile da calcolare. Se i tentativi sono minimi ($t = 1$) non c'è penalità, se invece lo studente

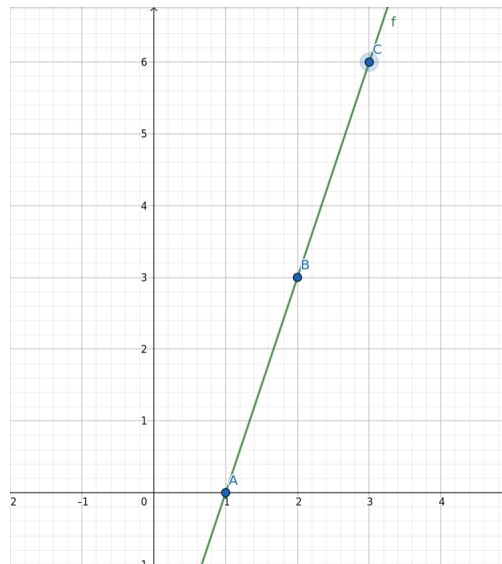


Fig. 3. Grafico della funzione $y = 3x - 3$

risponde nel doppio dei tentativi ($t = 2$) togliamo 3 punti fino a 6 punti per il triplo ($t = 3$) delle risposte necessarie.

Mettendo tutto assieme risulta:

$$v = 31 - \alpha \cdot \left(\#err + \frac{\#ext + \#plur}{2} + (3 \cdot t - 3) \right)$$

4 Generazione del linguaggio

4.1 Requisiti e codice

Alla classe `NaturalLanguageGenerator` sono necessari solo due componenti per funzionare: **il sentiment** e **il motore markoviano**. Andremo a specificare cosa serve a cosa

nel prossimo sotto-paragrafo. In se, il codice non è difficile da leggere: sono presenti solo due metodi abbastanza auto-esplicativi: il primo per leggere le frasi da un corpus scritto ed il secondo per generarle, tenendo conto del corpus stesso.

4.2 Generazione delle frasi

Le frasi vengono generate utilizzando delle **catene di Markov**, avendo come input un corpora di frasi, diviso per sentimento. Per farlo, è stata usata una classe chiamata **markovify**. Il funzionamento è veramente semplice: per ogni corpora viene costruita una catena di markov corrispondente, che si ripercorre per **100 volte** al fine di generare una nuova frase tenendo conto del corpus. Per differenziare la generazione, sono stati creati 6 corpus diversi: 3 per le domande (uno per sentimento) e 3 per le frasi filler (sempre uno per sentimento).

4.3 Funzionamento pratico

Dato il corpus giocattolo seguente, estratto dal nostro:

- (...)
- Is there Fluxweed in the potion? You can't make mistakes a this point of the exam.
- Do you remember if Chicken is in the potion? You have 2 seconds to think about it.
- Moving on. Do you need to put Valerian Spring in the potion?
- (...)

La catena di Markov estrae componenti che si ripetono spesso nelle frasi ed effettivamente li mischia, generando delle frasi del tipo: *"Is there Chicken in the potion? You can't make mistakes a this point of the exam."* oppure *"Moving on. Do you need to put Fluxweed in the potion? You have 2 seconds to think about it."*.

4.4 Errori di generazione

Ovviamente, più le frasi si somigliano, più sarà facile generare una frase di senso compiuto (dato che nella generazione non facciamo ragionamento di tipo semantico, ci possiamo aspettare molto spesso delle frasi senza un senso logico), del tipo: *"What more can you tell me more ingredients."* dato che dopo *"What more can you tell"* solitamente viene una frase con *'about'*, mentre esiste anche la frase *"can you tell me more ingredients?"*.

5 Conclusioni

5.1 Conversazione e Mood

Una parte fondamentale della conversazione che si ha con un assistente è sicuramente il fatto che esso non sia **statico** nel tempo ma che riesca ad interpretare la situazione e rispondere

di conseguenza. Questo fatto è stato codificato sotto forma di **emozioni dinamiche**.

Il metodo `check_for_mood_change` ci restituisce, data la situazione corrente del nostro frame, l'emozione corretta da includere nel nostro assistente personale. L'emozione cambia totalmente il corpus utilizzato da Piton, i voti che ci può dare e la quantità di errori che possiamo effettuare prima che l'interazione non si interrompa.

5.1.1 Da angry ad happy

```

Hello, I'm Professor Piton.
I can recognize an unprepared student right away,
I'll give you a chance this time, but let's hurry...
Give me all the ingredients and the quantities of the forgetfulness potion.
Hello. The first ingredient of the f.p. is Lether River Water.
You were not present at the laboratories...What are the odds that Knotgrass might be in the potion?
No, I know that this is a trick!
You... you really don't know what else is in the potion?
I know that Valerian Spring is the second ingredient known in the potion...
You... you really don't know what else is in there?
I think that the third one is the standard ingredient!
Come on, this is not enough, do you know the real ingredients or are you just here to waste my time?
The last one is mistletoe berry.
Your grade is 30...
Excellent job, my student. You are a genius!

```

Fig. 4. Interazione perfetta tra un Piton angry ed un ottimo studente

In questo primo caso possiamo notare dall'immagine 4 come un'interazione perfetta può trasformare anche l'emozione più bassa (angry) nella più alta (happy). Dallo stato angry, però, non potrà passare direttamente ad happy, ma dovrà essere un passaggio graduale, giusto per non renderlo esattamente uno *psicopatico* (un normale umano, se non in rari casi, non passa immediatamente dalla rabbia alla gioia).

5.1.2 Da happy ad angry

```

Hello, I'm Professor Piton.
hello welcome to the potion exam,
I've always seen you in class you will surely be prepared, let's start with this question...
As you will remember the training we did, the forgetfulness potion contains what ingredients?
I can give you an hint, if you want...
Can I ask you about wands?
The only person here that can make the questions is me.
I can give you an hint, if you want...
Ok, I think that cherries are in the potion
Do you think that Knotgrass is in the potion?
yes I do
That's okay but specify the other ingredients are present in this potion?
Another ingredient is Lether river water
That's okay but specify the other ingredients commonly found in the potion?
Maybe... spiders!
Is there a chance that the potion contain the Standard Ingredient?
yes, the standard ingredient is in there
I think we are not done, but we are not done, but we are almost there, tell me about the potion?
The last one is mistletoe berry.
Here we go, what other kind of ingredients are present in this potion?
I'm sorry, the last one was valerian spring!
Your grade is 22...
You passed. Get out. The cauldron is not for you.

```

Fig. 5. L'utente non sa molto: Piton passa abbastanza velocemente da essere felice ad arrabbiato

In questo secondo caso cerchiamo di far passare Piton dall'essere felice ad essere arrabbiato: l'unica maniera ovviamente è quella di sbagliare la maggior parte delle domande e di portare l'interazione ad avere quanti più turni possibili.

5.1.3 Un fallimento totale

```
Hello, I'm Professor Piton.  
Welcome to the potion exam,  
I will ask you some question about what i explained during the course  
Tell me the ingredients of the polyjuice potion.  
You should also tell me about the quantities... But it's up to you.  
The one I remember now is a cherry  
Is the Fluxweed used in this potion?  
No it is not  
Do you need to put Valerian Spring is in the potion?  
yes, it's probably in there...  
What are the other ingredients...  
Maybe... the standard ingredient  
What more can you tell me about the potion?. Specify the other ingredients might be in the potion?  
the head of a spider  
What are the odds that Knotgrass might be in the potion?  
No, it's not in there  
I think we are not done, tell me about the potion?. Specify the other ingredients are used to make this potion?  
I don't remember anything else... I'm sorry.  
You failed. Get out. The cauldron is not for you.
```

Fig. 6. Un fallimento totale da parte dell'utente. Ogni risposta è errata.

Come già citato in [3](#) il discorso termina immediatamente con un fallimento nel qual caso lo studente non sappia rispondere ad una domanda seriamente.

5.2 Possibili miglioramenti

5.2.1 Nell'analisi del discorso

Il nostro sistema di dialogo è in grado di esaminare ed elaborare un ampio set di tipologie di risposte, domande, frasi non utili; è comunque possibile approfondire in merito e integrare tutte le possibili tipologie di risposte che un utente può porre.

5.2.2 Nella generazione del linguaggio

Come spiegato nei paragrafi precedenti la generazione del linguaggio avviene tramite catene di Markov, è stata già discusso il problema delle possibili frasi prive di senso o comunque non utili al nostro scopo. Può essere sicuramente utile ampliare la lista di frasi utilizzate per la generazione. Ma una possibile soluzione definitiva può essere quella di utilizzare dei **transformer** come gpt3 per generare domande e frasi filler sempre corrette e coerenti.

5.2.3 Migliore Usabilità

L'interazione con terminale può risultare scomoda, soprattutto per persone fuori dal mondo informatico. Per migliorare l'usabilità del progetto si dovrebbe quindi usare un sistema di *speech-to-text* per riconoscere il parlato ed uno di *text-to-speech* per dar voce al professore e rendere il dialogo il più umano possibile.