220005783 | Theo Murton | CS1006 Eliza report | Tutor Simon Dobson

Overview

In the specification we were required to write a single java program that could take one of three scripts and respond in a human like way according to which script was used. We managed to do this as well as all the 'going further' elements of the specification, and also some others things we felt added to the chatbot.

Design

Before the program starts in earnest it has to read the files. I standardised the layout of my scripts with common headings such as KEYWORDS and MEMORY, and have a method that reads between each one and stores the lines in a list. Throughout the code we constantly use a tokenizer and Lists instead of strings to manage the input. We found this gave us much more power in altering word order, substituting etc.

The main format of the java program is a big while loop. This while loop continues so long as a boolean 'isQuit' is false.

On each iteration of the loop, standard input is received and converted into a space buffered String List to make it easier to analyse word by word.

The first checked thing is for a quit keyword. If one is found the program quits immediately with a randomly selected goodbye message.

If not, the first method applied is pre substitutions. The list of pre substitutions is of the format:

words to be swapped > words to be added
words to be swapped > words to be added
etc
etc

The program tells which words are which in the pre substitutions list by using the > sign as a delimiter. We made our own compareWords method so that the words to be swapped can be many words. We may have been able to do this with the Pattern and Matcher classes, but only found out about them in the last week when the engine was already complete and unpicking it would have been redundant when our solution worked in the same way (albeit in more lines). The substituter method is then used to remove and add words.

Next, keywords are checked for using the compare words method. This returns -1 if there is no match and otherwise returns the index of the first instance of the match. Again, the method works with multiple word phrases. The comparer works down the keyword list which is written in order of priority so be design the first match it finds will be the one with the highest priority.

If a match is found then the substituter method deals with the substitution of keywords and phrases. The scripts are formatted like this:

key phrase > format of decomposition < sentence () (the brackets indicate where the remembered words should go) | (this | symbol denotes the end of the recomposition rule.)

There can be several formats of decomposition. One is * (). This indicates everything before the keyphrase should be deleted and everything after remembered. () * indicates the opposite. This provides flexibility in decomposition: we can remember 'my cat' in 'my cat DIED' or we can remember 'sad' in 'I AM FEELING sad'. The | at the end of the line indicates the end of a recomposition rule. One decomposition can have as many recomposition rules as it wants: the recomposition is randomly selected. For reference, for similar key phrases like 'I feel' and 'feeling'; these are treated as separate phrases even though they contain the same word. For our purposes the grammatical structure of sentences is arguably even more important than their meaning: as in, 'feeling' has more in common with 'thinking' than it does 'I feel'. To facilitate this we implemented a synonym method. At the bottom of the keywords list is a list of synonyms, written like this:

key phrase > SYNONYM other key phrase | <

If one of these synonyms if found then the substituter method is used to substitute the synonym and the decompose method is called again within itself on the new complete phrase, which will be guaranteed a new 'proper' match on this run of the decompose method.

Recompositions may or may not contain a () to denote repeating the remembered text. For instance, the keyword 'yes' will likely not need a recall of input text, just a specific response.

If a keyword rule does have a (), once the recomposition is complete then the remembered fragment of the sentence is stored in a list of 'memories'. If not, this method is bypassed.

In any case, before the recomposition is conducted, the fragment is passed through the substituter method to apply the postsubstitutions which are formatted in the same way as the pre substitutions. These mostly swap personnage such as I and you.

If a keyword match is not found a randomiser is called. There is a 2/5 chance of:

A randomly selected 'dumb idle' response that is still in keeping with the personality of the script. For instance, the psychotherapist might say 'let's unpack that' as a generic response to most things. The current random value is stored and compared so that in the case of two consecutive non keyword matches and 2/5 picks, the same idle response is not repeated twice in a row as this would break immersion. This idle response is undoubtedly the most unhuman part of the program, but it is a necessity since we cannot exhaustively implement specific keywords.

There is a 3/5 chance of:

A 'memory' fragment from a previous keyword match being called. Each keyword rule has an associated recomposition memory rule. If the memory list is empty, then the program defaults to an idle response. Once a memory is recalled it is deleted from the list to avoid repetition. The memory format is:

(number of keyword line to refer to) > sentence (saved fragment)

Some of the lists the methods return have a final element which is not an input but a special integer. For instance, when a fragment is decomposed it has the line number of the keyword appended to the end so that the recall method knows where to look. This number is the last thing added to the list before the return, and is dealt with and stripped off immediately once the list is processed. We don't know if this is absolute best practice but it was the solution we came up with when we wanted to save calling methods multiple times and working around methods only being able to return 1 type of thing.

Up until this point everything has been dealt with using individual words and lists and a lot of iteration. The final step, regardless of what has happened before, is to clean up the sentence into one correctly cased and punctuated string using 'make final sentence' method. This is then printed out.

A few methods are of note from a design perspective. The most complicated one was subtituter: this was made tricky as I wanted it to be able to handle substitutions and additions of different lengths. For instance, swapping from 'cant' to 'can not' gains a word and vice versa for 'really like' to 'love'. Since I used the index match from the decomposition to indicate where the recomposition should take place, this word difference could have caused errors if the sentence contains more than one substitution. To handle this I had to track the differences and then calculate the total difference before an index and offset the insertion by this amount. Another thing I want to avoid was a substitution cycle: for instance, 'you are' being changed to 'I am', the loop repeating to check again, and finding 'I am' and changing it back to 'you are' ad infinitum. To manage this I implemented a 'banned set'- once a word (or more often words) have been substituted then their indexes are stored in a 'banned set'. When looking for multiple substitutions, the banned set is continually updated and these values are not considered in the next run.

For the third personality we decided to do yoda as he has a distinct pattern of speaking. The fact that the important part of the sentence comes first in his speak also allowed us to display the flexibility of our recomposition engine.

We believe we have ironed out most if not all of the bugs from our program: if passed a correctly formatted document then the program never generates exceptions. If no document argument is supplied then an error message is generated and the program is quit. If the file given cannot be found then the same happens with a relevant message. If another error is generated, it is assumed that the file has a formatting error and a relevant message is generated and the program is quit.

The main logical error with the program is in the limitations of the keyword and the grammar they attempt to implement. However, we believe that these problems are fairly unsolvable given the time we have. For instance, should 'you' post substitute to 'I' or 'me' (nominative or accusative)? Most native English speakers would struggle to explain the difference, so explaining it to a computer within the restraints of the simple Eliza program would be nigh on impossible. We decided to go with the heuristic solution of 'me' in this example as it makes more sense more often in Eliza conversations. The Eliza illusion is generally fairly convincing but occasionally Eliza will recall something you have input much later in the conversation. This will be grammatically correct and might actually happen in a real life conversation, but can be a bit surprising and highlights the fact that Eliza is far from perfect! The 'idle' phrases are generally vague enough to be convincing but occasionally an input and an idle response will be very dissonant, but there's not much we can do about this without adding keywords exhaustively.

We managed to implement all the going further features. The yet undiscussed multiple sentence feature turned out to be easy to implement. Since we only implement one keyword response and there is a strict priority as well, the 'keyword comparison' hinted at in the specification is done by default anyway. Then, the only next step was breaking the loop in the decomposition method if a punctuation .,?! was encountered which suggests a new sentence. This is possible for both the 'remember before' and 'remember after' decomposition methods.

We should add a couple of design choices may seem odd: for instance, we made a whole 'compare words' method instead of using Pattern and Matcher, and wrote our script as a unique readable format when JSON may have been more conventional. The reason for both of these was we only found out about both of these at the end of week 3 when the engine was already complete. We saw little point going back and unpicking our correctly working methods.

Some extra features we added included idiot-proofing the input- if you look in the examples we often omit apostrophes from common abbreviations like 'can't', frequently used the incorrect case (words like 'I'), and are inconsistent with punctuation at the end of sentences. We made a few methods a bit more complex (make tokens, make final sentence, get input) to make sure keywords could always be detected and a grammatically correct response was supplied.

Sample dialogues
psychotherapist.txt

Salutations. Pray tell me of your struggles.

im devastated, my mum died yesterday.

I'm so sorry that your mum died.

i love her

That's positive, it's good to love her.

not now it isn't, i feel awful

Can it be bad to feel awful?

of course

Since your mum died, what have you done?

i have thought about her every day

Does it concern you that you have thought about her every day?

maybe. i don't know what to do.

Sometimes it is impossible to know what to do.

i cant help feeling guilty about it!

You say you are feeling guilty about it?

should i?

It's for you to decide if you should.

thanks a lot.

Great, looks like we're making progress.

yoda.txt

tm282@trenco:~/Documents/CS1006/Eliza $ java Chatbot 8yo.txt

Mum says I shouldn't talk to strangers.

i think that she is a very wise person

Why do you think she is a very wise person?

because she gave you good advice

My teacher also thinks that she is a very wise person.

do you like school?

Of course i like school!

i thought maybe you wouldn't given you're 8.

Maybe shmaybe!

now you're just speaking nonsense.

No i'm not!

you are.

No i'm not!

fine, have it your way. i don't know what to say now.

Why don't you know what to say now?

you have exhausted my imagination

I was thinking, and i don't know what to say now either.

how does that work?

What do you mean?

Nothing.

I said i really liked school!

yes, you made that very clear

You seem sure.

i should be

No.

what?

What do you mean?

forget it. do you know what pizza is?

What do you think? am i the sort of boy who looks like i know what pizza is?

i thought so

Did you really think i know what pizza is?

no, my mum didn't let me have pizza until i was 10

No no no no no!

you can still have pizza

What do you mean?

that's enough. Bye

Nice talking.

tm282@trenco:~/Documents/CS1006/Eliza $ java Chatbot yoda.txt

Expect to see you, I did not.

then you have a limited imagination

Naturally, a limited imagination i have.

you are so humble

So humble, i am not.

how paradoxical. now, i want to be a jedi

Manage your desires, you must.

i think that that was a bit rude.

Think that was a bit rude, why do you?

because you aren't respecting my dreams

To understand better, please help me.

i like lightsabers, the force and everything

Like lightsabers, i do as well.

should i get one?

Your own path, you must decide.

fine. do you know obi-wan? i am a huge fan of his

Know obi-wan, do i? think on this i must.

really? do you really know that many jedi?

Really know that many jedi, do i? think that of me, do you?

Yes

Sure you are, it would seem.

i am sure

Patient, you must be.

ok. do you like star wars?

Still considering my own preferences on star wars, i am.

i love star wars.

Only in moderation must we love star wars.

i hate star wars.

To hate star wars, the path to the dark side is.

are you serious?

Serious, i am. know this, how did you?

ha ha. so all star wars fans are sith?

Destroy the sith, we must.

great. i am feeling up to it

Feeling up to it, why are you?

because. anyway, bye

Disappeared, another few minutes of my life have. Yess. Mhhhhhhhm.

Link to repository by path:
/cs/groups/cs1006-p1-group16/Eliza

Link to repository by URL:
https://github.com/theomurton/Eliza

Report is 1897 words excluding sample dialogues.