**Nishad Aherrao**

**Objective:**

To create a prolog database with data enough to replicate a chat bot that acts intelligently. Main purpose of the bot is to converse with a child who is back from school and enquire about his overall day what he did what he didn’t and if he had fun doing it.

**Challenges:**

Main challenges in this were deciding on a language for making the GUI as some were high level and some were lower level but didn’t have enough documentation or guides for help. I ended up going with python as I was already comfortable using python. I used the module/library called **pyswip**. It acts as a bridge in between prolog and python.

Other than that to tackle the duplicate activities for example draw can be under do() or play() with different followups and to make sure they are set to positive liking and not asked again.

**SETUP:**

Install SWIprolog, Python and pyswip library fort this app to run.

Usefull commands for bash:

sudo apt install swi-prolog

pip install pyswip

sudo apt-get install python3.6

If path not set then may need to run these two commands too I had to do this:

export PATH=$PATH:PATH=$PATH:/Applications/SWI-Prolog.app/Contents/swipl/bin/x86\_64-darwin15.6.0

export DYLD\_FALLBACK\_LIBRARY\_PATH=/Applications/SWI-Prolog.app/Contents/swipl/lib/x86\_64-darwin15.6.0

**Explanation of code:**

I have used the **easygui** library to make simple dialogue boxes without defining color or height and just use them for the yes and no input.

The **random** library is used to get a random entry from a list and is used in the positive and negative reactions as well as for the followup question when there are multiple followups.

Moving on to the Reactions I have two lists one positive and other negative. I use the positive reaction and the start of the sentence when asking the question if the child responded positively to the previous question.

For ex. If the child says he had fun my reply starts with Wow or Nice etc. same for the negative part.

In the future I could see I had to ask questions again and again one for the initial and one for the followup so I defined two templates named as query for the first question and followUp for the followup. These get used in the future their parameters are what get used in changing the question around and not hardcoding a lot of stuff thus keeping the program more generic.

We need a starting point like an ice breaker or a conversation starter to start the program off so I defaulted this to playground one of the values in database.The counter turns true only when we are done with all entries so it keeps the program looping till all data is traversed.

If this is the first time the question is being asked we have a little custom greeting ‘Welcome Back!’.

After this the program now asks a question and if he gets a positive response then asks a followup question or else it moves on to something else as it reckons that the child did not have fun in that activity. The program will move from play() to do() also if anyone of the play() activities was not liked by the child and comeback to it later.

Basically the program thinks if he didn’t have fun playing playground theres a good chance he didn’t have fun playing anything else in that category and moves to eat or do or travel and comes back to playing later.

Also theres a snippet to take care of the duplicate activities occurring in multiple category and making sure they are not asked twice once for each category, instead they get asked just once. The followup is taken at random so may be different in a second run.

Because of the pyswip I was able to use assertz to add to the database while running the program through python and gathering the input through python. So I was not only able to read the prolog db but also write to it. I assert or add the positives to the like list and negatives to dislike list after listening to the input and also keep a track of whats asked and whats not in the history list. Also finally use the len() func to help determine when to terminate the prog. (at length 0).

**Explaination of the prolog database**

I have five categories of activities with a number of sub activities and with some occurring multiple times as build can be something a child do() as well as play().

The following are my explanation of each category with relatable definitions.

*pFollow(find\_it\_fun).*

*eFollow(eat\_it\_all).*

*dFollow(find\_it\_helpful).*

*sFollow(find\_it\_beautiful).*

*aFollow(get\_punishment).*

I initialize the counters or storing lists before we start.

I have 3 ask rules and the first extra one is to start us off consider it a default one.

The actual first rule is to start by asking a question about something the child likes , Is related and not been asked before.

The second one is about a random question which is totally unrelated.

The last one helps us pick a random one like we did the playground one only this time it wont be hardcoded it will be picked at random by the AI.

So actually you need only 3 but the extra one is for start.

We have our standardized rules to append lists together which we use in the related and followup rules to find the relation between X and Y.

The related rule shows us relevant options to choose a relatable question to the previous one.

The followup rule as it says helps us select a followup relation.

The random rule gives us a random list.

**Prolog Database Code**

/\*

This is the things to ask related to each action category.

\*/

play([playground,seasaw,toys,alphabet,hideandseek,sandbox,playmat,cars,trains,slides,build]).

eat([cake, toffee, candy, sandwich, pizza, cheerios, veggies, fries, noodles, meat]).

do([build, read, draw, dance,hideandseek,alphabet]).

see([river, well, farm, city,draw]).

argue([argue\_with\_friends,argue\_with\_teachers]).

/\*

The followup questions for each category to be asked.

\*/

pFollow(find\_it\_fun).

eFollow(eat\_it\_all).

dFollow(find\_it\_helpful).

sFollow(find\_it\_beautiful).

aFollow(get\_punishment).

/\*

Initialize all for multiple runs

\*/

like(nothing).

dislike(nothing)..

history(nothing).

/\*

Ask something to start off the system as everything is initialized at the start.

\*/

ask(playground, 0).

/\*

As we already asked a random item we now ask a related item.

\*/

ask(X,Y):-like(Y), related(X,Y), \+ history(X).

/\*

Also keep track that we don't repeat them.

\*/

ask(X,Y):- random(X), \+ related(X,Y), \+ history(X).

/\*

Now we ask a random one after the followup to start again

\*/

ask(X,Y):- random(X), \+ history(X).

/\*

Rule to append lists and add to it

\*/

append([X | Y], Z, [X | W]) :- append(Y, Z, W).

append([], X, X).

/\*

See the relationship in between the two X and Y

\*/

related(X,Y) :-

play(L), member(X, L), member(Y, L);

eat(L), member(X, L), member(Y, L);

do(L), member(X, L), member(Y, L);

see(L), member(X, L), member(Y, L);

argue(L), member(X, L), member(Y, L).

/\*

X and Y relationship in between member and the followup category we declared earlier for each action

\*/

followup(X, Y) :-

play(L), member(X, L), pFollow(Y);

eat(L), member(X, L), eFollow(Y);

do(L), member(X, L), dFollow(Y);

see(L), member(X, L),sFollow(Y);

argue(L), member(X, L), aFollow(Y).

/\*

Random

\*/

random(X) :-

play(L), member(X, L);

eat(L), member(X, L);

do(L), member(X, L);

see(L), member(X, L);

argue(L), member(X, L).

**Python GUI Interface Code**

from pyswip import \*

import easygui

import random

# Creating a positive and negative expression list that will be used at random based on input to make the robot human like.

posReaction = ['Good boy!','Wow','Nice!','Makes sense.','Cool']

negReaction = ['Really?','Its fine','Oh!','Well its in the past now.']

#defining a default query structure/template to use everytime in the if else loops below.

#For the first question/ Random new Question

def query(initial, expression1):

# Using easygui library/module to make a simple gui application to gather input in yes no format

return easygui.ynbox('{} Was there {} today'.format(expression1, initial), '', ('Yes', 'No'))

#For the followup question.

def followUp(initial):

#Using easygui library/module to make a simple gui application to gather input in yes no format

return easygui.ynbox('Did you {}?'.format(initial), '', ('Yes', 'No'))

# This is the main logic that makes use of the previous defined templates and prolog database.

def main():

# Found this defaut initialization on google and also the code to consult prolog file.

prolog = Prolog()

prolog.consult("logic.pl")

# Need to start with some topic as everything is empty now.

initial = 'playground'

# This is the counter that keeps track if all the options have been asked or not.

counter = False

# Checks if this is the first time the program is being run.

first = True

pos = True

# logic program loop

while not counter:

if first:

# If this is the first time ask the default question and turn first to false so we dont get stuck in this loop.

first = False

answer = query(initial, 'Welcome back!')

else:

if positive:

# Ask the question with a random positive starting emotional response.

answer = query(initial, random.choice(posReaction))

else:

# Ask the question with a random negative starting emotional response.

answer = query(initial, random.choice(negReaction))

if answer:

# Only ask a followup question if the child actually enjoyed doing the previous activity.

followup = list(prolog.query('followup({}, Y)'.format(initial)))

# There can be multiple followup like for build,draw,alphabet,hideandseek etc.

followupRand = random.choice(followup)['Y']

followupPos = followUp(followupRand)

# IF they like the followup we add that to the like list too keep track and not repeat.

if followupPos:

prolog.assertz('like({})'.format(initial))

positive = True

else:

prolog.assertz('dislike({})'.format(initial))

positive = False

else:

# Default to dislike if they didnt take part in the activity as we assume they didnt like it

prolog.assertz('dislike({})'.format(initial))

positive = False

# Keep a track of what has been asked

prolog.assertz('history({})'.format(initial))

queryResult = list(prolog.query('ask(X, {})'.format(initial)))

# Prolog function to find the length

if len(queryResult) == 0:

# Length reaches 0 means no more data left

end=easygui.ynbox('So overall was it a fun day!')

if end:

easygui.msgbox('Nice now have some rest you did good!')

else:

easygui.msgbox('Aw its ok baby lets go shopping and eat Ice Cream!')

counter = True

break

initial = queryResult[0]['X']

# Indent it back to the left to end the main

if \_\_name\_\_ == "\_\_main\_\_":

main()