



**NANYANG**  
**TECHNOLOGICAL**  
**UNIVERSITY**

**CZ3005 Artificial Intelligence**  
**Assignment 2**

**Lab Group: SSP2**  
**Name: Chen Cheng**  
**Matriculation ID: U1722345A**

## Assignment 2: Kid's day at school

### Assignment Description:

Assume that the prolog script is a parent, trying to know about a kid's day at school. The prolog script should converse intelligently with the kid as follows. The prolog script should ask a question to the kid that kid can answer yes or no. Depending on whether the answer is yes or no, prolog script should ask a related question or another random question. For example, if the kid says yes to whether it ate or not, the query can be a food item, or whether the kid used fork or spoon, or whether the kid washed hand. Similarly, related to games.

### Overview

In this assignment, I have implemented a Prolog-based interact to imitate the conversation between a parent and a child. The child answers the parent and the parent ask a series of questions based on the kid's response.

The implementation is divided into 2 parts: the core Prolog script and a GUI created with the help of a python package: "easygui". The file "knowledge\_base.pl" stores the core Prolog script which represents the Knowledge-Base of the parent. The file "interface.py" is the python script that provides GUI for this conversation.

### Package Used:

- pyswip

This package provides a bridge between Python and Prolog, it consists functions such as "prolog.query()" and "prolog.assertz()" which can pass the queries generated from Python script to the Prolog logic.

- easygui

This package provides a simple and convenient way to create a Graphical User Interface, user's response with the GUI is passed back as function returns.

- random

This package is mainly used to generate random prefixes and suffixes so that the program sound more robot like.

However, though `pyswip` provides a quick and easy connection between Python and Prolog, it only works with Prolog below version 8.0. Hence, downgrade of Prolog is necessary for the program to work properly.

### Model: `knowledge_base.pl`

The first part of the predicates provides a way to retrieve elements from stored lists. With the help of three helper predicates: `related(X, Y)`, `random(X)` and `history(X)`, we could make sure that related elements will be returned first before the program goes on to search random element. Also, as `history(<some constant>)` will be asserted every time after it is returned by the program, we could avoid getting duplicate elements as the return result.

```
/* setting up basic ask query to retrieve on activity*/
ask(X, Y):-
    related(X, Y), \+ history(X).
ask(X, Y):-
    random(X), \+ related(X, Y), \+ history(X).
```

The following lists are also added to the Prolog script.

```
/* the list of activities the kid could do in school */
play([slides, sandbox, toys, trains, cars, playmat, ball]).
eat([cake, toffee, candy, sandwich, pizza, cheerios, veggies, fries, hamburgers, chocolate]).
see([bird, flowers]).
learn([alphabet, numbers, mathematics, drawing]).

/* the list of possible positive feelings the kid could have */
play_positive_description([fun, enjoyable, exciting]).
eat_positive_description([tasty, delicious, healthy]).
see_positive_description([colourful, fancy, nice]).
learn_positive_description([enjoyable, fun, interesting]).

/* the list of possible negative feelings the kid could have */
play_negative_description([tiring, exhausting, bad]).
eat_negative_description([awful, bitter, unsavory]).
see_negative_description([dull, ugly, bad-looking]).
learn_negative_description([difficult, frustrating]).

/* the list of possible actions the kid should have done */
play_action([wash_hand_after_that, drink_enough_water]).
eat_action([clean_the_plate, clean_the_table, wash_hands_before_that]).
see_action([try_to_take_photo, touch_it]).
learn_action([take_notes, pay_attention, ask_teacher_questions]).
```

Besides having the basic lists under each activity, I also added a list of description and actions. This is to add more interactions between the program and the user as well as to make the program more interesting.

The following diagram shows some of the helper predicates that help to retrieve items in the list according to appropriate context. For example, calling `followup_positive_description(X, cake)`. will only give you description to describe food.

```
/* define the relationship between two related elements*/
related(X,Y) :-
    play(L), member(X, L), member(Y, L);
    eat(L), member(X, L), member(Y, L);
    see(L), member(X, L), member(Y, L);
    learn(L), member(X, L), member(Y, L).

/* Define one postive description of a particular element*/
followup_positive_description(X, Y):-
    play(L), member(X, L), play_positive_description(S), member(Y, S) ;
    eat(L), member(X, L), eat_positive_description(S), member(Y, S);
    see(L), member(X, L), see_positive_description(S), member(Y, S);
    learn(L), member(X, L), learn_positive_description(S), member(Y, S).

/* Define one negative description associated with a particular element */
followup_negative_description(X, Y):-
    play(L), member(X, L), play_negative_description(S), member(Y, S);
    eat(L), member(X, L), eat_negative_description(S), member(Y, S);
    see(L), member(X, L), see_negative_description(S), member(Y, S);
    learn(L), member(X, L), learn_negative_description(S), member(Y, S).

/* Define one action associated with a particular element */
followup_action(X, Y):-
    play(L), member(X, L), play_action(S), member(Y, S);
    eat(L), member(X, L), eat_action(S), member(Y, S);
    see(L), member(X, L), see_action(S), member(Y, S);
    learn(L), member(X, L), learn_action(S), member(Y, S).
```

## Python Script: interface.py

Below is the main function loop inside the python script. The script first asks a neutral question about the first item in the “play” list, following questions regarding the activity the kid has done in school will adapt to his response in previous questions.

To add more interaction between the program and the kid, if the kid has done one particular activity, the program will go on to ask about the kid’s feeling towards that activity as well as whether he has done the actions he is supposed to do.

The loop stops when there is no activities returned by Prolog queries, and a closing message will be displayed.

```
def main():
    activity_item = "slides"
    positive_feedback = True
    has_done = ask(activity_item, "Welcome back kiddo, how is school today?", "?")
    while True:
        if has_done:
            # ask if the kid has positive feeling towards the activity.
            ask_about_description(activity_item)
            # ask if the kid has done correct actions after the activity
            ask_about_action(activity_item)
        else:
            # marking that negative feedback was received.
            positive_feedback = False

        prolog.assertz("history({})".format(activity_item))
        next_items = list(prolog.query('ask(X, {})'.format(activity_item)))
        if len(next_items) == 0:
            break
        activity_item = next_items[0]['X']
        # react differently according to previous feedback
        has_done = ask_about_activity(positive_feedback, activity_item)

    # sending closing message to the user
    easygui.msgbox("That's all, gald we had this conversation!")
```

## Partial Sample of Generated Dialogue

Kid's day at school.

Welcome back kiddo, how is school today? Was there slides today ?

the kid clicked "Yes"

Yes

No

Kid's day at school.

Can you describe it? Was it enjoyable ?

the kid clicked "Yes"

Yes

No

It is nice to hear that!

OK

Kid's day at school.

Oh, by the way, Did you wash hand after that ?

the kid clicked "Yes"

Yes No

Kid's day at school.

Was there cars today also?

the kid clicked "Yes"

Yes No

Kid's day at school.

That's nice! Did you drink enough water as well?

the kid clicked "No"

Yes No

Kid's day at school.

Nevermind then. Was there playmat today instead?

Yes

No