CZ3005 Artificial Intelligence
Report on Lab 2
Patient with a Sympathetic Doctor

Niu Haoyu[1], Group TSP4
N1902565A

11 November 2019

[1] N1902565A@e.ntu.edu.sg

I. Introduction

In this lab, we are required to implement a Prolog based AI to interact with users and to give results dynamically. To improve my understanding of Prolog, I chose the fourth assignment, the sympathetic doctor. Here, the system needs to pretend itself as a doctor, trying to diagnose what illness the patient may have who can only say yes or no.

Users play as patients in this game and answer yes or no to each question asked by the doctor. After finishing all possible symptoms, the AI doctor makes a diagnosis based on the positive symptoms described by the patient and the knowledge stored in his database to determine what illnesses the patient may have.

In this program, I have set 5 different levels of pain (unbearable pain, lot of pain, manageable pain, mild pain, no pain), 5 different kinds of mood (calm, angry, weepy, anxious, malaise), 9 different symptoms (temperature, sweat, ache, sneeze, cough, blood, breathe rapidly, headache, bruise), and 6 possible diagnoses including no illness, fever, cold, injury, cancer, and anxiety disorder, with each diagnosis characterized by 5 or more symptoms.

When diagnosing, the doctor first asks users about their pain level and mood, and then ask whether they have each of the 9 symptoms mentioned above to get the eventual diagnosis. During the whole process, the doctor dynamically adjusts his gestures and tones according to choices made by patients.

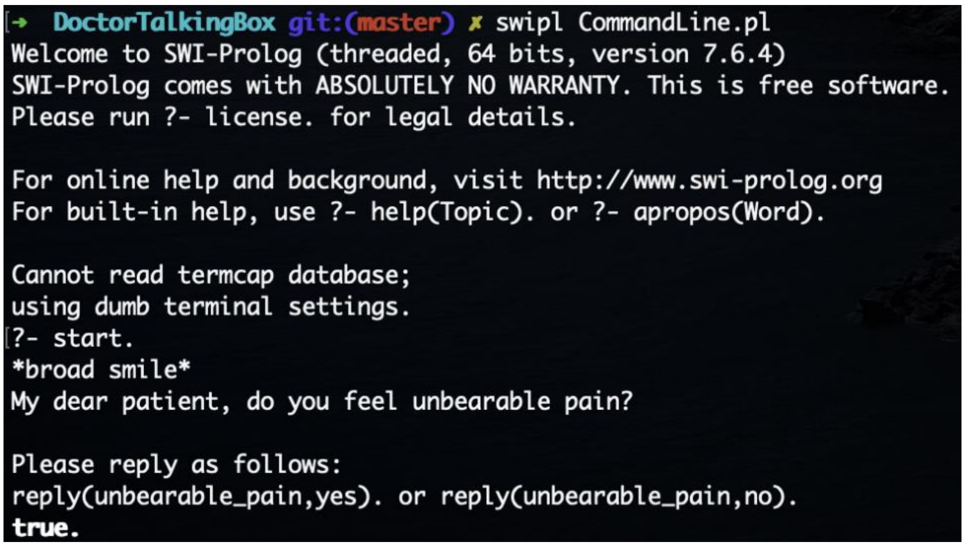
This submission also includes a GUI version of the sympathetic doctor, which is supported by a HTTP web server constructed using Prolog and its built-in libraries only.

II. User's Guidance

Before running this program, please make sure that your computer has installed the latest version of SWI-Prolog.

First, you need to switch to the main directory which has `CommandLine.pl` and `Server.pl`.

Then, to start the command line version, please run `swipl CommandLine.pl` in your terminal, and then enter `?- halt.` to start. For the web GUI version, please run `swipl Server.pl` in your terminal, and then access http://127.0.0.1:8000/ from your browsers to start.



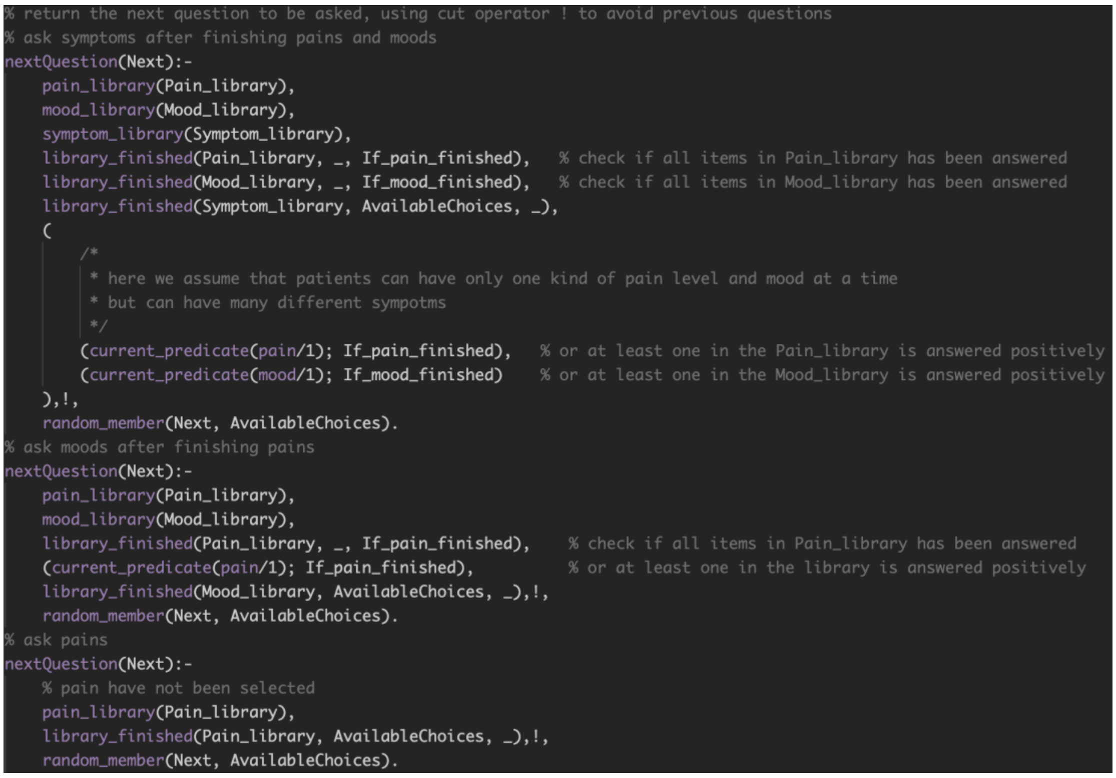
```
[→  DoctorTalkingBox git:(master) ✗ swipl CommandLine.pl
Welcome to SWI-Prolog (threaded, 64 bits, version 7.6.4)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit http://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

Cannot read termcap database;
using dumb terminal settings.
[?- start.
*broad smile*
My dear patient, do you feel unbearable pain?

Please reply as follows:
reply(unbearable_pain,yes). or reply(unbearable_pain,no).
true.
```

Figure 1: Example of Starting via Command Line

If you would like to try again after the doctor gives his diagnosis, you may stop the `swipl` environment by enter `?- halt.` and then start the AI doctor again to test another group of symptoms.

III. Implementations

To make this program more reusable, I separate the implementations of the AI doctor into two parts: core logic and user interaction.

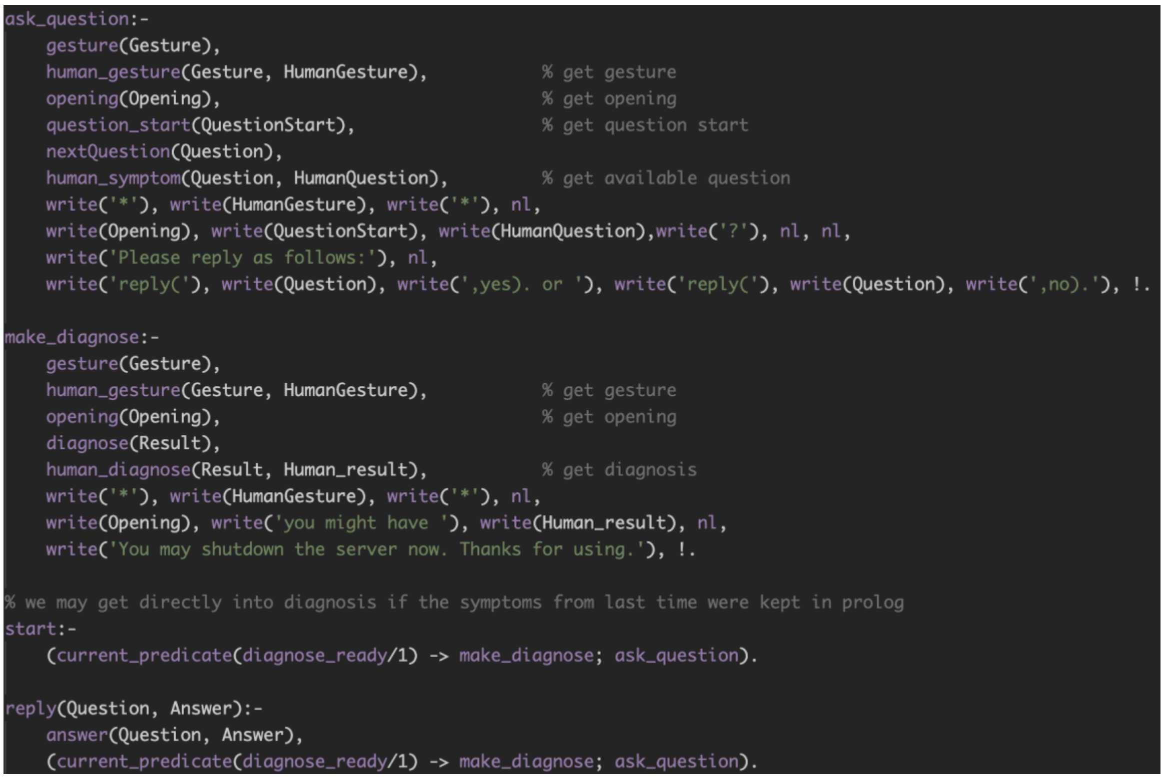


```
% return the next question to be asked, using cut operator ! to avoid previous questions
% ask symptoms after finishing pains and moods
nextQuestion(Next):-
    pain_library(Pain_library),
    mood_library(Mood_library),
    symptom_library(Symptom_library),
    library_finished(Pain_library, _, If_pain_finished),   % check if all items in Pain_library has been answered
    library_finished(Mood_library, _, If_mood_finished),   % check if all items in Mood_library has been answered
    library_finished(Symptom_library, AvailableChoices, _),
    (
        /*
         * here we assume that patients can have only one kind of pain level and mood at a time
         * but can have many different sympotms
         */
        (current_predicate(pain/1); If_pain_finished),   % or at least one in the Pain_library is answered positively
        (current_predicate(mood/1); If_mood_finished)    % or at least one in the Mood_library is answered positively
    ),!,
    random_member(Next, AvailableChoices).
% ask moods after finishing pains
nextQuestion(Next):-
    pain_library(Pain_library),
    mood_library(Mood_library),
    library_finished(Pain_library, _, If_pain_finished),    % check if all items in Pain_library has been answered
    (current_predicate(pain/1); If_pain_finished),          % or at least one in the library is answered positively
    library_finished(Mood_library, AvailableChoices, _),!,
    random_member(Next, AvailableChoices).
% ask pains
nextQuestion(Next):-
    % pain have not been selected
    pain_library(Pain_library),
    library_finished(Pain_library, AvailableChoices, _),!,
    random_member(Next, AvailableChoices).
```

Figure 2: Part of Code in `DoctorLogic.pl` to Select Next Question Dynamically

DoctorLogic.pl takes a response to store the “intelligence” of the virtual doctor and to provide an API towards the user interaction part. All libraries of symptoms, gestures, patient’s choices, and their corresponding processing logic are defined and stored in this file. Specifically, the doctor asks the patient’s pain level at the very beginning and store the answer inside the memory, which is selected to perform because the program finds that there is no answer regarding previous questions stored in memory. Then similarly, the doctor asks the patient’s mood and other symptoms since the program finds that the patient has answered his/her pain level with mood or some symptoms in the library unanswered. Moreover, this file is also responsible for selecting the doctor’s gestures and speaking patterns based on the patient’s previous answers, enabling the doctor’s behavior to be as appropriate as possible. Notice that for better interpretability, the program does a manual translation when giving output to outside interaction part.

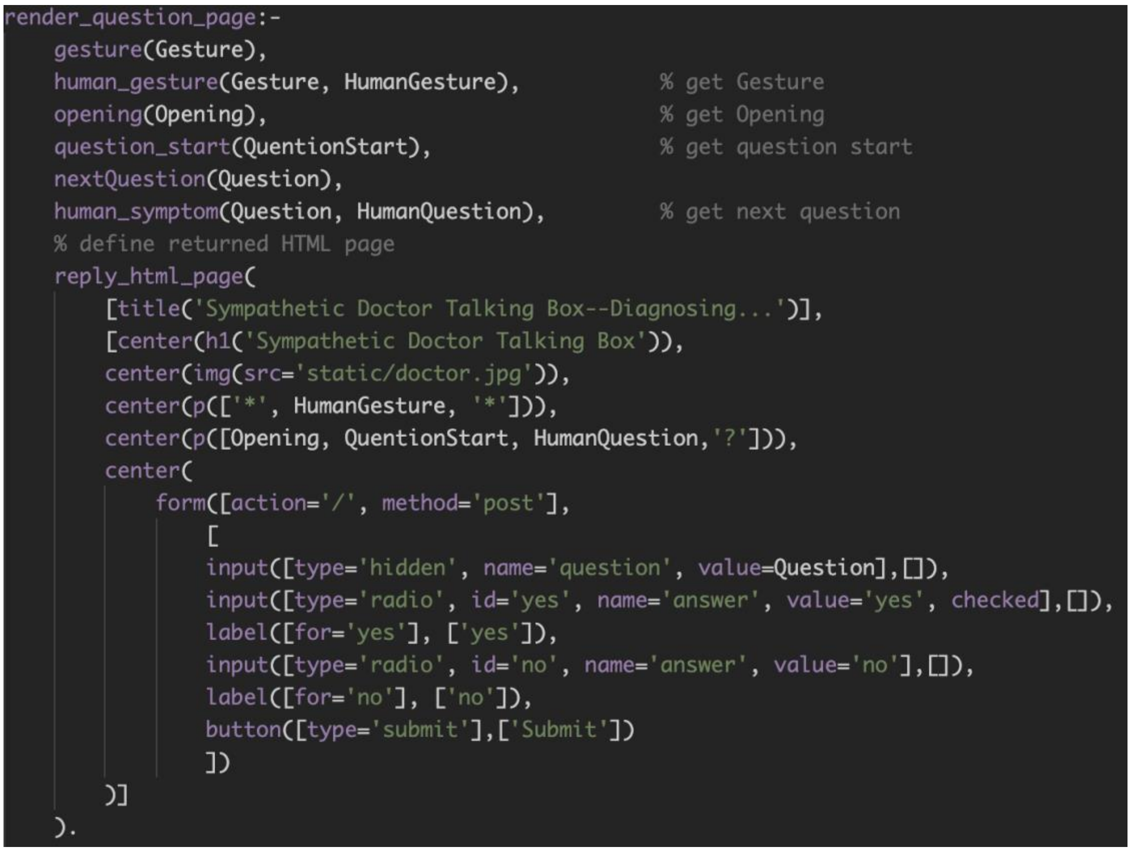
CommandLine.pl is responsible for enabling users to interact with the Prolog AI doctor via the command-line interface. Code in this file receives predicates entered by users, call predicates defined in DoctorLogic.pl for further analysis, and making responses by retrieving the next question to ask from DoctorLogic.pl.



```prolog
ask_question:-
    gesture(Gesture),
    human_gesture(Gesture, HumanGesture),          % get gesture
    opening(Opening),                              % get opening
    question_start(QuestionStart),                 % get question start
    nextQuestion(Question),
    human_symptom(Question, HumanQuestion),        % get available question
    write('*'), write(HumanGesture), write('*'), nl,
    write(Opening), write(QuestionStart), write(HumanQuestion),write('?'), nl, nl,
    write('Please reply as follows:'), nl,
    write('reply('), write(Question), write(',yes). or '), write('reply('), write(Question), write(',no).'), !.

make_diagnose:-
    gesture(Gesture),
    human_gesture(Gesture, HumanGesture),          % get gesture
    opening(Opening),                              % get opening
    diagnose(Result),
    human_diagnose(Result, Human_result),          % get diagnosis
    write('*'), write(HumanGesture), write('*'), nl,
    write(Opening), write('you might have '), write(Human_result), nl,
    write('You may shutdown the server now. Thanks for using.'), !.

% we may get directly into diagnosis if the symptoms from last time were kept in prolog
start:-
    (current_predicate(diagnose_ready/1) -> make_diagnose; ask_question).

reply(Question, Answer):-
    answer(Question, Answer),
    (current_predicate(diagnose_ready/1) -> make_diagnose; ask_question).
```

Figure 3: Code in CommandLine.pl *to interact with users*

Last but not least, Server.pl is responsible for constructing a simple HTTP server using Prolog and its built-in libraries only, allowing users to interact in GUI with the AI doctor through any web browser. Specifically, the server keeps listening on the localhost:8000 port after it starts and returns the initial HTML welcome page for any incoming GET request, which is generally the AI doctor asking the patient about his/her pain level. After a user clicks the “submit” button on the displayed page, a form including the user’s answer and its corresponding question are sent to the server by a POST request. Hence, the server extracts this information from any incoming POST request, then sends it to DoctorLogic.pl for processing, and eventually gets the next question or diagnosis from `DoctorLogic.pl` to render a HTML page and to make a response to where the POST request comes from.



```
render_question_page:-
    gesture(Gesture),
    human_gesture(Gesture, HumanGesture),           % get Gesture
    opening(Opening),                               % get Opening
    question_start(QuentionStart),                  % get question start
    nextQuestion(Question),
    human_symptom(Question, HumanQuestion),         % get next question
    % define returned HTML page
    reply_html_page(
        [title('Sympathetic Doctor Talking Box--Diagnosing...')],
        [center(h1('Sympathetic Doctor Talking Box')),
        center(img(src='static/doctor.jpg')),
        center(p(['*', HumanGesture, '*'])),
        center(p([Opening, QuentionStart, HumanQuestion,'?'])),
        center(
            form([action='/', method='post'],
                [
                input([type='hidden', name='question', value=Question],[]),
                input([type='radio', id='yes', name='answer', value='yes', checked],[]),
                label([for='yes'], ['yes']),
                input([type='radio', id='no', name='answer', value='no'],[]),
                label([for='no'], ['no']),
                button([type='submit'],['Submit'])
                ])
        )]
    ).
```

Figure 4: Part of Code in `Server.pl` *to Render a Question Page*

One thing I would like to mention in particular is that since the Prolog language is gradually fading out nowadays, there are fewer and fewer developers who are still willing to maintain interfaces between Prolog and some modern programming languages like Python and JavaScript (e.g., pyswip). This directly led me to encounter several bugs (some of them are even quite obvious and easy to fix) within interfaces when I was trying to build the GUI using the latest version of these two languages and their libraries, which is the reason why I finally chose to use the original Prolog for web development. Fortunately, I finally made it with tutorials and instructions over the Internet.

IV. Conclusion

Thanks to adequately designed architecture in this program, now it is quite easy for us to add some symptoms, diseases, even methods of user interaction like apps by Java or MATLAB into existing knowledge library with little modification to the logic part. Moreover, the server implemented using Prolog makes this project easy to use on various platforms without the need to install those annoying runtime environments. We can even add some CSS and simple JavaScript to the web page to make it look better in the future.

This is all for this report. For further details, please refer to the attached source code, which has plenty of comments inside.

For any questions, please contact N1902565A@e.ntu.edu.sg or niuhaoyu17@gmail.com.

Thanks for reading!