AI DOCTOR IN PROLOG

Mini Project
Submitted in partial fulfillment of the requirement of University of Mumbai
For the Degree of

(Computer Engineering)

By

1) Soham Chakraborty	**ID No: TU3F1920046**
2) Mohit Hitendra Bisht	**ID No: TU3F1920055**
3) Shubhankar Kanoje	**ID No: TU3F1920056**

Under the Guidance of

Prof.Rohini Patil

Department of Computer Engineering
TERNA ENGINEERING COLLEGE
Plot no.12, Sector-22, Opp. Nerul Railway station,
Phase-11, Nerul (w), Navi Mumbai 400706
UNIVERSITY OF MUMBAI

Terna Engineering College

NERUL, NAVI MUMBAI

CERTIFICATE

This is to certify that

1) Soham Chakraborty	**ID No: TU3F1920046**
2) Mohit Hitendra Bisht	**ID No: TU3F1920055**
3) Shubhankar Kanoje	**ID No: TU3F1920056**

Has satisfactorily completed the requirements of the ***Mini Project***

Of subject

Artificial Intelligence

As prescribed by the ***University of Mumbai*** *Under the guidance of*

Prof. Rohini Patil

Subject Incharge **HOD**

Index

Chapter 1
Introduction

The report describes a program written in SWI-prolog declarative language to mimic the diagnosis process by conversing with a patient who can only say yes or no, and additional two commands to quit the program and force to diagnose.

The doctor will ask about the mood and pain level of the patient, and the tone of the questions will change accordingly. Afterwards the doctor will ask consecutive questions to determine the symptoms the patient has. If the patient affirms one particular symptom, the symptom will be recorded, and another related symptom will be randomly picked, and the program will ask the patient again. Otherwise if the patient negates the symptom, the program will record it as well and randomly pick one symptom from all available unasked symptom poll to ask the patient.

If one or more diseased can be diagnosed, the program will display the diagnose analysis and terminate. If no match can be found, the program will print the error message and terminate. The diseases and their related symptoms are pre-defined and stored in the program's knowledge base. The program can be rerun for arbitrary times in one execution process with previous diagnosis history cleared.

Chapter 2

Problem Statement

To create an empathetic and comprehensive medical system using prolog which can ask the patient about a variety of their symptoms and use appropriate facial expressions and gestures to make the patient feel comfortable and diagnose the patient efficiently.

2.1. Predefined Knowledge Base

2.1.1. Mood and Pain List

```
/*Below are the default data of mood, pain, and disease symptoms.*/
moodlist([calm, tense, fearful, gloomy, melancholy]).
painlist(['no pain', 'mild pain', 'moderate pain', 'severe pain',
'intolerable pain']).

/*Rate the seriousness of the mood and pain, with 1 being the least
serious and 5 being the most serious*/
mood(5, melancholy).
mood(4, fearful).
mood(3, gloomy).
mood(2, tense).
mood(1, calm).
pain(5, 'intolerable pain').
pain(4, 'severe pain').
pain(3, 'mild pain').
pain(2, 'moderate pain').
pain(1, 'no pain').
```

5 types of mood and 5 levels of pain are chosen and each of the mood and pain are associated with a seriousness degree. The higher the degree, the more serious and mentally and physically unstable the patient is. The degrees will function as a mathematical model in later predicates to determine the tone of the doctor. In our program, we take x = 0.2*mood + 0.8*pain to be the overall seriousness of the patient.

2.1.2. Doctor's Gesture List

```
norgesture(['broad_smile', 'joke', 'beaming_voice']).
polgesture(['looks concerned','mellow voice','light
touch','faint_smile']).
calgesture(['greet', 'look_composed', 'look attentive']).
```

The above facts model the possible tones the doctor might express in the diagnosis process. Depending on the seriousness degree, the doctor will choose one gesture in either one of the three gesture lists. For example, if the overall serious is higher than 3, the doctor might initiate a question as below:

```
Doctor: * looks concerned *

: Do you sneeze ? y/n/q/d:
```

2.1.2. Diseases and Symptoms List:

```
cold(['sneeze', 'cough', 'have congestion', 'have mild headache',
'have stuffy nose']).
diabetes(['feel very thirsty', 'have extreme fatigue', 'have weight
loss', 'have blurry vision', 'urinate often']).
hiv(['have muscle aches', 'have sore throat', 'have fatigue', 'have
swollen lymph nodes', 'have mouth ulcers']).
lungcancer(['cough blood', 'have chest pain', 'have hoarseness',
'lose appetite', 'have shortness of breath']).
rabies(['feel easily irritated', 'have hallucinations', 'experience
excessive movements', 'have extreme sensitivity to bright lights',
'have convulsions']).
```

5 diseases which reside relatively far away from each other in the human disease spectrum are modelled with their symptoms. Two symptoms under one disease are considered related. The following predicate is written to determine the “relationship” between two symptoms:

```
/*Y is related to X if they are both symptoms under one disease*/
related(X, Y):-
        cold(L), member(X, L), member(Y, L);
        diabetes(L), member(X, L), member(Y, L);
        hiv(L), member(X, L), member(Y, L);
        lungcancer(L), member(X, L), member(Y, L);
        rabies(L), member(X, L), member(Y, L).
```

2.1.3. Dynamic Knowledge Base and Initialization

```
/*Initialization*/
askedsymp([]).
has([]).
hasnot([]).

/*Default patient mood and patient pain*/
patientmood(calm).
patientpain('no pain').
```

askedsymp([]) initially contains one empty list. Along with the diagnosis process, all asked symptoms will be updated here to prevent from being asked again.

Similar to askedsymp([]), has([]) stores all symptoms the patient confirms and hasnot([]) stores all symptoms the patient negates. The list in has() will be utilized to diagnose the patient. The list in hasnot() has no practical use in the program, but can be easily extended for more advanced expansion.

The default mood and pain level for patient are “calm” and “no pain”. These will be updated when the diagnosis process begins.

2.2. Diagnosis Workflow

2.2.1. Begin and Initialize

```
begin():-
        retractall(has(_)),
        assert(has([])),
        retractall(hasnot(_)),
        assert(hasnot([])),
        retractall(askedsymp(_)),
        assert(askedsymp([])),
        format('Doctor: ~n Welcome to the sympathetic doctor, we will
begin the diagnose process now. ~nWe will ask you a few questions
about your cnditions and symptoms. ~nIf you feel like you have
provided enough information, enter \'d\' to diagnose. ~nOtherwise,
enter \'q\' to abort the session ~n'),
        moodlist(L),
        askMood(L).
```

Enter the goal **begin().** to start the program. This predicate clears the information of the previous patient and initialize has(), hasnot(), and askedsymp(). A welcome message will be printed. Then it gets the moodList L and enter the predicate askMood(L) with L.

2.2.1. Get Mood from Patient

```
askMood(L):-
        sympatheticAsk(),
        list_empty(L, V),
        (V == true -> format('Unfortuanetely we could not diagnose
you if you do not provide credible answers to our questions. Try
again later. ~n');
         L = [H|T]
        ),
        format('May I ask do you have a ~w mood today? y/n/q/d: ~n',
[H]),
        read(AnswerMood),
        (AnswerMood == q -> abort;
         AnswerMood == y -> diagnoseMood(H);
         AnswerMood == n -> askMood(T);
         AnswerMood == d -> diagnosebegin()).

/*This clears the default patientmood and records the user input.*/
diagnoseMood(M):- retractall(patientmood(_)), assert(patientmood(M)),
painlist(L), askPain(L).
```

SympatheticAsk() is a procedure to determine the tone based on the overall seriousness of the patient. See section 2.3. for details.

List_empty(L, V) is a procedure to determine if the list is empty. If L is empty, V will be true, and vice versa. See section 2.4. for details.

The procedure will firstly choose a tone for the doctor, and if L is empty, terminate the program since all mood choices have run out. Otherwise, the head of L (call it H) will be asked. If patient has mood H, diagnoseMood(H) will update patientmood(), and get painlist(L) and enter procedure askPain(L). If not, the procedure recursively calls itself with the rest of the mood list.

If user enters ‘q’, the program will be forced to terminate. And if user enters ‘d’, the program will be forced to diagnose.

2.2.2. Get Pain from Patient

```
askPain(L):-
        sympatheticAsk(),
        list_empty(L, V),
        (V == true -> format('Sorry we could not diagnose you if you
do not provide credible answer to our questions. Try again later.
~n');
         L = [H|T]
         ),
        format('May I ask do you have ~w today? y/n/q/d: ~n', [H]),
        read(AnswerPain),
        (AnswerPain == q -> abort;
         AnswerPain == y -> diagnosePain(H);
         AnswerPain == n -> askPain(T);
         AnswerPain == d -> diagnosebegin()).

diagnosePain(P):- retractall(patientpain(_)), assert(patientpain(P)),
askSymptom('cough').
```

The mechanism is similar to askMood. askPain(L) will update patientpain according to user input. And will ask the user if he/she has “cough” to begin the askSymptom(S) procedure.

2.2.3. Ask Symptoms of Patient

```
askSymptom(X):-
        sympatheticAsk(),
        format("Do you ~w ? y/n/q/d: ~n", [X]),
        read(AnswerSymptom),
        (AnswerSymptom == q ->abort();
         AnswerSymptom == y -> affirmSymptom(X);
         AnswerSymptom == n -> negateSymptom(X);
         AnswerSymptom == d -> diagnosebegin()
         ).
```

SympatheticAsk() will be called first. Depending on the user input, either affirmSymptom or negateSymptopm will be called.

2.2.4. Affirmation and Negation of Symptom

```
affirmSymptom(X):-
        has(H), append(H, [X], HX), assert(has(HX)), retract(has(H)),
        askedsymp(L), append(L, [X], LX), assert(askedsymp(LX)),
retract(askedsymp(L)),
        findall(Y,(related(X, Y), askedsymp(N), \+member(Y, N)), R),
        (R == [] -> diagnosebegin();
         random_member(Q, R)),
        askSymptom(Q).

negateSymptom(X):-
        hasnot(N), append(N, [X], NX), assert(hasnot(NX)),
retract(hasnot(N)),
        askedsymp(L), append(L, [X], LX), assert(askedsymp(LX)),
retract(askedsymp(L)),
        cold(C), diabetes(D), hiv(H), lungcancer(K), rabies(B),
        append(C, D, CD), append(CD, H, CDH), append(CDH, K, CDHK),
append(CDHK, B, CDHKB),
        askedsymp(S),
        findall(Y,(member(Y,CDHKB), \+member(Y, S)), R),
        (R == [] -> diagnosebegin();
        random_member(Q, R)),
        askSymptom(Q).
```

If the patient affirms a symptom, it will be updated in has() and askedsymp() respectively, then a list that contains all symptoms related to X into a list Y. A random symptom in Y will be asked, if Y is not empty. Otherwise it means all symptoms of one disease have been depleted and the diagnosis will begin.

If the patient affirms a symptom, it will be updated in has() and askedsymp() respectively, then a list that contains all symptoms from all diseases will be generated. A random symptom from the all symptom pool and is not a member of asked symptoms (R) will be picked and asked. If R is empty, it means all symptoms have been asked and the diagnose will begin.

2.2.5. Diagnose Session

```
diagnoseBegin():-
        sympatheticAsk(),
        format('Thank you! We have completed the diagnosis
process.~nFollowing is our analysis: ~n'),
        diagnose().

diagnose():-
        diagnosecold();
        diagnosediabetes();
        diagnosehiv();
        diagnoselungcancer();
        diagnoserabies();
        format('We could not draw any conclusion based on the
information you provided. Please try asgain later.~n'),
        diagnoseEnd().

diagnoseEnd():-
        format("Thank you for using our service! You can consult an-
other disease now. ~n"),
        abort().
```

diagnoseBegin() will print a message and enter diagnose procedure.

diagnose() will determine the disease the patient has by calling individual diagnose procedure and enter diagnoseEnd().

diagnoseEnd() will print a message and abort the session.

2.2.6. Disease Diagnose

Take diagnosecold(). as an example:

```
diagnosecold():-
        has(H),cold(C), intersection(H, C, R), length(R, L),
        (L >= 4 -> format('you might have cold.~n')),
        diagnoseEnd().
```

Procedures of this kind will get the list in has() and intersect with all the symptoms a disease has. If more than 4 symptoms are matched, the diagnosis will be printed and the diagnose session would end.

2.3. Sympathetic Gesture Choosing.

```
sympatheticAsk():-
        format('Doctor: '),
        patientmood(M),
        mood(D, M),
        patientpain(P),
        pain(S, P),
        chooseTone(D*0.2+S*0.8).

chooseTone(D):-
        (D=<1 -> normalGesture();
         D=<3 -> politeGesture();
         D=<5 -> calmingGesture()
        ),
        format(': ').

normalGesture():- norgesture(L), random_member(X, L), format('* ~w
*~n', [X]).
politeGesture():- polgesture(L), random_member(X, L), format('* ~w
*~n', [X]).
calmingGesture():- calgesture(L), random_member(X, L), format('* ~w
*~n', [X]).
```

Procedure sympatheticAsk() will get the seriousness degree of the patient and pass the overall seriousness (D*0.2+S*0.8) to chooseTone(D). Depending on D, one of the gesture procedure will be called and one gesture will be chosen for the doctor.

2.4. Utilization Predicates

```
list_empty([], true).
list_empty([_|_], false).
```

To check if a given list is empty.

Chapter 3

Implementation

1. Prolog code:

```
/*Mark the following knowledge base as dynamic to modify them in
the program*/
:- dynamic(askedsymp/1).
:- dynamic(has/1).
:- dynamic(hasnot/1).
:- dynamic(patientmood/1).
:- dynamic(patientpain/1).

/*The begin predicate serves as an initialization. It will remove
the past diagnosis history and print out a welcome message, then
continue to the procedure askMood to record the patient/'s mood*/
begin():-
     retractall(has(_)),
     assert(has([])),
     retractall(hasnot(_)),
     assert(hasnot([])),
     retractall(askedsymp(_)),
     assert(askedsymp([])),
     format('Doctor: ~n Welcome to the sympathetic doctor, we will
begin the diagnose process now. ~nWe will ask you a few questions
about your conditions and symptoms. ~nIf you feel like you have
provided enough information, enter \'d\' to diagnose. ~nOtherwise,
enter \'q\' to abort the session ~n'),
     moodlist(L),
     askMood(L).

/*This procedure gets the current patient mood and patient pain,
use a weighted sum of mood and pain and pass the result to
chooseTone() to intellectually choose a appropriate tone for the
doctor.
Here I have set the default formula to D*0.2+S*0.8 as the
seriousness indicator, but the fractions are free to change if
necessary. Higher the value, more concerned the doctor would be.
*/
sympatheticAsk():-
     format('Doctor: '),
     patientmood(M),
     mood(D, M),
     patientpain(P),
     pain(S, P),
     chooseTone(D*0.2+S*0.8).

/*Based on the argument D, which is the seriousness indicator, one
of the Gestures will be called.*/
chooseTone(D):-
     (D=<1 -> normalGesture();
      D=<3 -> politeGesture();
      D=<5 -> calmingGesture()
     ),
     format(': ').

/*Gestures of a kind is randomly chosen*/
normalGesture():- norgesture(L), random_member(X, L), format('* ~w
*~n', [X]).
politeGesture():- polgesture(L), random_member(X, L), format('* ~w
*~n', [X]).
calmingGesture():- calgesture(L), random_member(X, L), format('* ~w
*~n', [X]).

/*Doctor speaks with a tone. If mood list has run out, terminate
the progam. Else, according to the answer of the patient, the
program either aborts, enters diagnose mood if answer is y,
recursively ask the next mood if answer is no, or begin
diagnosis.*/
askMood(L):-
     sympatheticAsk(),
     list_empty(L, V),
     (V == true -> format('Unfortuanetely we could not diagnose you
if you do not provide credible answers to our questions. Try again
later. ~n');
      L = [H|T]
     ),
     format('May I ask do you have a ~w mood today? y/n/q/d: ~n',
[H]),
     read(AnswerMood),
     (AnswerMood == q -> abort;
      AnswerMood == y -> diagnoseMood(H);
      AnswerMood == n -> askMood(T);
      AnswerMood == d -> diagnosebegin()).

/*This clears the default patientmood and records the user input.*/
diagnoseMood(M):- retractall(patientmood(_)),
assert(patientmood(M)), painlist(L), askPain(L).

/*Similar to askMood*/
```

```
askPain(L):-
     sympatheticAsk(),
     list_empty(L, V),
     (V == true -> format('Sorry we could not diagnose you if you
do not provide credible answer to our questions. Try again later.
~n');
      L = [H|T]
      ),
     format('May I ask do you have ~w today? y/n/q/d: ~n', [H]),
     read(AnswerPain),
     (AnswerPain == q -> abort;
      AnswerPain == y -> diagnosePain(H);
      AnswerPain == n -> askPain(T);
      AnswerPain == d -> diagnosebegin()).

/*Similar to diagnoseMood, this clears the default patient pain
which is no pain and records the user input. Additionally this
enters next phase askSymptom.*/
diagnosePain(P):- retractall(patientpain(_)),
assert(patientpain(P)), askSymptom('cough').

/*Similar to askMood. If answer is y, enters affirmSymptom, and if
answer if n, enters negateSymptom*/
askSymptom(X):-
     sympatheticAsk(),
     format("Do you ~w ? y/n/q/d: ~n", [X]),
     read(AnswerSymptom),
     (AnswerSymptom == q ->abort();
      AnswerSymptom == y -> affirmSymptom(X);
      AnswerSymptom == n -> negateSymptom(X);
      AnswerSymptom == d -> diagnosebegin()
      ).

/*Entering this procedure means the patient has confirmed a symptom.
has() will be updated and askedsymp(), which contains all asked
symptoms will be updated as well.
Based on the symptom, a related symptom will be asked. If all
symptoms of a given disease has been confirmed by the patient, end
asking and enter diagnose procedure.*/
affirmSymptom(X):-
     has(H), append(H, [X], HX), assert(has(HX)), retract(has(H)),
     askedsymp(L), append(L, [X], LX), assert(askedsymp(LX)),
retract(askedsymp(L)),
     findall(Y,(related(X, Y), askedsymp(N), \+member(Y, N)), R),
     (R == [] -> diagnosebegin();
      random_member(Q, R)),
     askSymptom(Q).
```

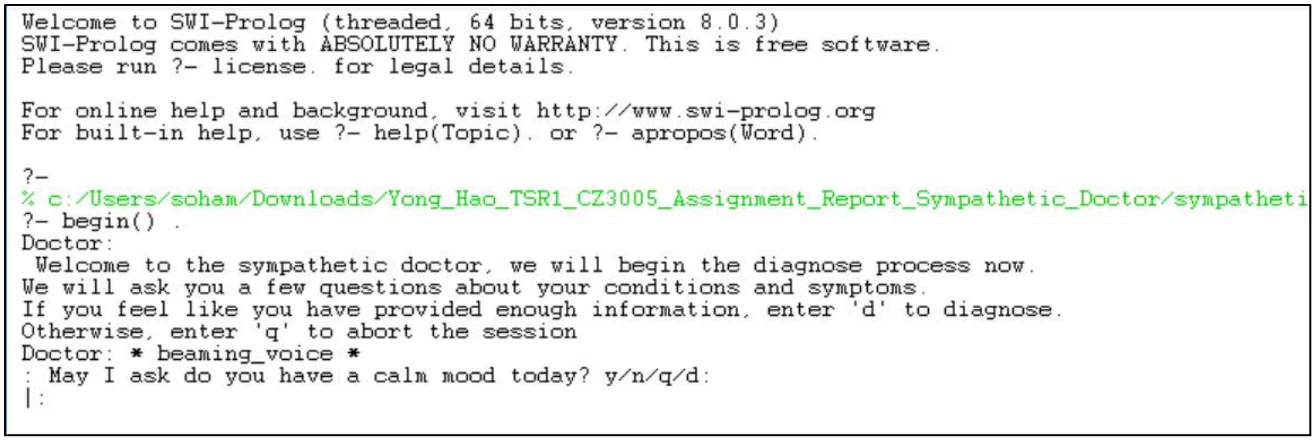
```
/*Similar to affirmSymptom, but all effects are negative. hasnot()
will be updated and askedsymp() will be updated as well. A random
unasked symptom from the symptom poll will be asked.
If no more symptoms are unasked, enter the diagnose procedure.*/
negateSymptom(X):-
     hasnot(N), append(N, [X], NX), assert(hasnot(NX)),
retract(hasnot(N)),
     askedsymp(L), append(L, [X], LX), assert(askedsymp(LX)),
retract(askedsymp(L)),
     cold(C), diabetes(D), hiv(H), lungcancer(K), rabies(B),
     append(C, D, CD), append(CD, H, CDH), append(CDH, K, CDHK),
append(CDHK, B, CDHKB),
     askedsymp(S),
     findall(Y,(member(Y,CDHKB), \+member(Y, S)), R),
     (R == [] -> diagnosebegin();
     random_member(Q, R)),
     askSymptom(Q).

/*Begin diagnose. Print a message.*/
diagnosebegin():-
     sympatheticAsk(),
     format('Thank you! We have completed the diagnosis
process.~nFollowing is our analysis: ~n'),
     diagnose().

/*Diagnose if either or all of the five diseases are diagnosed.*/
diagnose():-
     diagnosecold();
     diagnosediabetes();
     diagnosehiv();
     diagnoselungcancer();
     diagnoserabies();
     format('We could not draw any conclusion based on the
information you provided. Please try asgain later.~n'),
     diagnoseEnd().


/*The following five clauses will compare the symptoms in has() and
the symptoms a disease possess, and if more than 4 symptoms are
matched, a diagnosis message will be printed.*/
diagnosecold():-
     has(H),cold(C), intersection(H, C, R), length(R, L),
     (L >= 4 -> format('you might have cold.~n')),
     diagnoseEnd().

diagnosediabetes():-
```

```
     has(H),diabetes(C), intersection(H, C, R), length(R, L),
     (L >= 4 -> format('you might have diabetes.~n')),
     diagnoseEnd().

diagnosehiv():-
     has(H),hiv(C), intersection(H, C, R), length(R, L),
     (L >= 4 -> format('you might have HIV.~n')),
     diagnoseEnd().

diagnoselungcancer():-
     has(H),lungcancer(C), intersection(H, C, R), length(R, L),
     (L >= 4 -> format('you might have lung cancer.~n')),
     diagnoseEnd().

diagnoserabies():-
     has(H),rabies(C), intersection(H, C, R), length(R, L),
     (L >= 4 -> format('you might have rabies.~n')),
     diagnoseEnd().

/*End of program*/
diagnoseEnd():-
     format("Thank you for using our service! You can consult
another disease now. ~n"),
     abort().

/*Y is related to X if they are both symptoms under one disease*/
related(X, Y):-
     cold(L), member(X, L), member(Y, L);
     diabetes(L), member(X, L), member(Y, L);
     hiv(L), member(X, L), member(Y, L);
     lungcancer(L), member(X, L), member(Y, L);
     rabies(L), member(X, L), member(Y, L).

/*Randomly choose a symptom from the poll of symptoms*/
randomSymptom(X):-
     cold(C), diabetes(D), hiv(H), lungcancer(L), rabies(R),
     append(C, D, CD), append(CD, H, CDH), append(CDH, L, CDHL),
append(CDHL, R, CDHLR),
     random_member(X, CDHLR).

/*A procedure to check if the list is empty*/
list_empty([], true).
list_empty([_|_], false).


/*Below are the default data of mood, pain, and disease symptoms.*/
moodlist([calm, tense, fearful, gloomy, melancholic]).
```

```
painlist(['no pain', 'mild pain', 'moderate pain', 'severe pain',
'intolerable pain']).

norgesture(['broad_smile', 'joke', 'beaming_voice']).
polgesture(['looks concerned','mellow voice','light
touch','faint_smile']).
calgesture(['greet', 'look_composed', 'look attentive']).

cold(['sneeze', 'cough', 'have congestion', 'have mild headache',
'have stuffy nose']).
diabetes(['feel very thirsty', 'have extreme fatigue', 'have weight
loss', 'have blurry vision', 'urinate often']).
hiv(['have muscle aches', 'have sore throat', 'have fatigue', 'have
swollen lymph nodes', 'have mouth ulcers']).
lungcancer(['cough blood', 'have chest pain', 'have hoarseness',
'lose appetite', 'have shortness of breath']).
rabies(['feel easily irritated', 'have hallucinations', 'experience
excessive movements', 'have extreme sensitivity to bright lights',
'have convulsions']).

/*Rate the seriousness of the mood and pain, with 1 being the least
serious and 5 being the most serious*/
mood(5, melancholic).
mood(4, fearful).
mood(3, gloomy).
mood(2, tense).
mood(1, calm).
pain(5, 'intolerable pain').
pain(4, 'severe pain').
pain(3, 'mild pain').
pain(2, 'moderate pain').
pain(1, 'no pain').

/*Initialization*/
askedsymp([]).
has([]).
hasnot([]).

/*Default patient mood and patient pain*/
patientmood(calm).
patientpain('no pain').
```

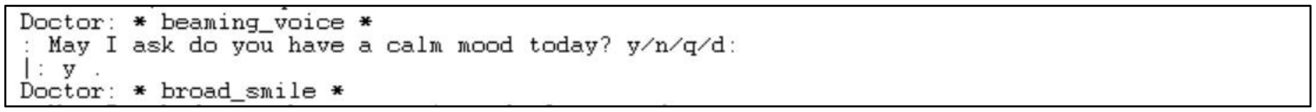
2. Program execution snapshots :

2.1 Start of execution



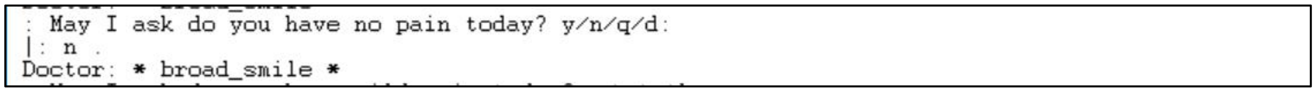
```
Welcome to SWI-Prolog (threaded, 64 bits, version 8.0.3)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

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

?-
% c:/Users/soham/Downloads/Yong_Hao_TSR1_CZ3005_Assignment_Report_Sympathetic_Doctor/sympatheti
?- begin() .
Doctor:
 Welcome to the sympathetic doctor, we will begin the diagnose process now.
We will ask you a few questions about your conditions and symptoms.
If you feel like you have provided enough information, enter 'd' to diagnose.
Otherwise, enter 'q' to abort the session
Doctor: * beaming_voice *
: May I ask do you have a calm mood today? y/n/q/d:
|:
```

2.2 responding to a question with "yes"



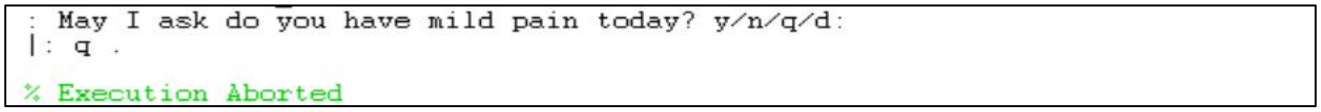
```
Doctor: * beaming_voice *
: May I ask do you have a calm mood today? y/n/q/d:
|: y .
Doctor: * broad_smile *
```

2.3 responding to a question with "no"



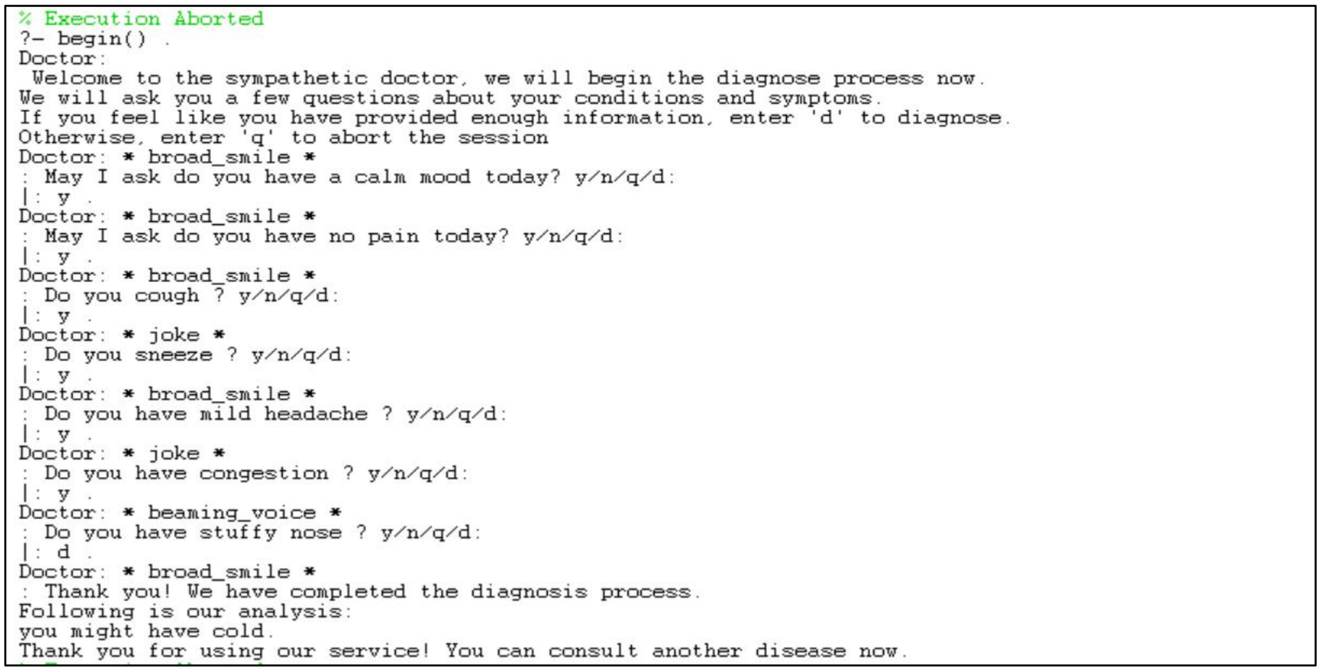
```
: May I ask do you have no pain today? y/n/q/d:
|: n .
Doctor: * broad_smile *
```

2.4 responding to a question with "abort session"



```
: May I ask do you have mild pain today? y/n/q/d:
|: q .

% Execution Aborted
```

2.5 responding to question with "instant diagnose"



```
% Execution Aborted
?- begin() .
Doctor:
 Welcome to the sympathetic doctor, we will begin the diagnose process now.
We will ask you a few questions about your conditions and symptoms.
If you feel like you have provided enough information, enter 'd' to diagnose.
Otherwise, enter 'q' to abort the session
Doctor: * broad_smile *
: May I ask do you have a calm mood today? y/n/q/d:
|: y .
Doctor: * broad_smile *
: May I ask do you have no pain today? y/n/q/d:
|: y .
Doctor: * broad_smile *
: Do you cough ? y/n/q/d:
|: y .
Doctor: * joke *
: Do you sneeze ? y/n/q/d:
|: y .
Doctor: * broad_smile *
: Do you have mild headache ? y/n/q/d:
|: y .
Doctor: * joke *
: Do you have congestion ? y/n/q/d:
|: y .
Doctor: * beaming_voice *
: Do you have stuffy nose ? y/n/q/d:
|: d .
Doctor: * broad_smile *
: Thank you! We have completed the diagnosis process.
Following is our analysis:
you might have cold.
Thank you for using our service! You can consult another disease now.
```

<u>Chapter 4</u>

Conclusion

In conclusion an artificial intelligence based doctor program was created in prolog which intelligently diagnosed patients while choosing the correct expressions and gestures while interacting with the patient.

Creation of a fully working program allowed the opportunity of working with a large number of different features and their implementations provided within Prolog.

Also working on this project as a team gave us a close to real world experience of working within software development teams and how members within the teams must work to streamline the production process and make debugging a friction free activity.

We achieved the expected output of creating an artificial intelligence based doctor emulator which was able to interact with patients in more human and empathetic way thus taking us a step closer to the major goal of creating completely automated healthcare system.

References

❖ **Websites**

1. https://www.swi-prolog.org/pldoc/doc_for?object=manual

2. https://www.youtube.com/watch?v=SykxWpFwMGs

3. https://stackoverflow.com/questions/55196608/or-operator-expected-after-expression-in-gnu-prolog

❖ **Books**

1. Adventure in prolog, 24th March 2017 by Dennis Merrit

~*~*~*~*~*~*~