Learning Outcome

The Chatbot project is a venue for students to achieve the learning outcomes below:

- LO1. Design and evaluate informed search algorithms and knowledge representations for problem solving
- LO2. Collaboratively build systems that consider a number of paths or strategies in order to improve its performance in achieving its goal in less amount of computing time, or by some other metric of performance
- LO3. Collaboratively build and evaluate an expert system prototype that addresses a need of a particular user group, community, or organization
- LO5. Articulate ideas and present results in correct technical written and oral English

Students will demonstrate their skills in modeling, developing and evaluating the knowledge base of their chatbot/expert system. They are going to analyze how the knowledge of the experts in the domain of their choice can be represented. Then, they are going to develop the chatbot and evaluate its intelligence. They will also write a report explaining strengths and weaknesses of the chatbot.

Medical Diagnostic Chatbot

The task is to develop a small medical diagnostic expert system for a list of diseases of your choice that is suitable for poor communities in the Philippines. Part of the system is a question-and-answer interface and a backward-chaining expert system shell.

Note that this domain (diagnosis of diseases that are prevalent in rural or poor communities) may be replaced by some other application domain of similar relevance and complexity.

The system should involve at least 15 individual diseases that are common among the poor or rural communities in the Philippines.

The expert system may start with rules or clauses that deal with general questions to establish the "chief complaint" and the "history of present illness (HPI)". The HPI interface would then be used to determine which specific medical diagnosis system to launch, which would differ depending on the so called "chief complaint".

You may need to augment the HPI rules with specific diagnostic rules once the HPI branches out to specific medical domains (e.g. infections in the upper respiratory tract, throat/tonsils, mouth and nose).

For the diagnosis portion of the system, you may use a backward-chaining expert system shell, such as the one found on page 6, which is written in Prolog.

Instructions

1. Form a group composed of maximum of 4 members.
2. Download Prolog from https://www.swi-prolog.org/
3. Select the 15 diseases that you want your chatbot to be an expert in diagnosing.
4. Build the knowledge base of your chatbot.
   a. The first option is to interview an expert. The expert from whom you will acquire knowledge for the expert system must normally have worked in his/her field of specialization for at least 5 years, ideally at least 10 years. Ideally, he/she is a GP, family medicine expert, or an internist and has had experience working in depressed and rural communities in the country.
   b. The second option is to review at least 5 related literature or websites. Using the same example, you must look for websites or other literatures giving information about the description,

symptoms, treatment, etc. of common illnesses. You might also want to look for the frequently asked questions regarding common illnesses. Please do note that the websites or literature must be credible enough to be the basis of your knowledge base.

c. In the diagnosis part of the Expert System, you cannot not require (nor refer to) results from expensive laboratory tests, such as ultra sound, x-ray, MRI, CT scan, and specific laboratory and microscopy tests.
d. In general, the medical diagnosis may only be based on basic patient information like age, gender, height, weight, and BP, as well as results from various probes using a stethoscope, visual inspection of an affected area, pressing of certain areas for numbness or pain, interview, and the like.
e. All these "tests and probes" should be assumed to be performed at some ill-equipped barangay health center or rural health unit, conducted by a nurse or midwife, with instructions from a doctor who is "virtually" present via the Internet and some Android device to personally interview the patient.
f. The diagnosis may involve information not coming from interview questions, e.g. with the aid of a glossary of pictures available to the nurse/midwife, such as the following:
    - "if the skin rashes look like the rashes depicted in picture 375, then chicken pox"
    - "if the back of the throat, when seen with a flashlight and tongue depressor, has spots of dark yellow like the ones depicted in picture 122, then advanced staphyloccus infection"
    - "if the wound area looks like the wound in picture 410, then gangrene"
g. The diagnosis may involve information that can be deduced from a stethoscope when placed on specific areas of the chest, back, or stomach cavity areas.
h. If the patient is in some emergency situation where there is no time, nor possibility, to seek medical care from a more advanced medical facility, the system should attempt to do a diagnosis, with a certainty factor.
i. If the patient is not in an emergency situation and the doctor would absolutely need specific tests procedures from a large medical facility (e.g. hospital) before a diagnosis can be made, then the diagnosis is "refer to a large medical facility".

5. Develop the chatbot using Prolog.
6. Write a documentation of your chatbot and the whole process of developing it.

Final Deliverables

The Gamebot project has two components – a type-written report and a demo of the bot.

1. Prepare a type-written report, **minimum of two pages** (excluding the title page), single-spaced, short-bond paper, containing the following:

I. Introduction

- Describe the domain (e.g., meningitis).
- Describe the task (e.g., diagnosis of meningitis) within the domain that you would like to automate.
- Explain why the task requires expertise (and 10 years to build this expertise), and what a person needs to do to develop expertise in this domain and task.
- Explain the significance of this expert system to a particular group, community, or organization.

II. Knowledge Base

- List and number the rules in your knowledge base in IF <antecedent> THEN <consequent> form.
- Explain the parts of the knowledge that were the easiest to finalize and that caused the least problems, and why these were so. Use figures for illustration.
- Explain the parts of the knowledge base that were the most difficult to finalize or caused the most problems, why these were so, and how they were revised. Use figures for illustration.

III. Results and Analysis

- Give at least five most impressive sample conversations with your knowledge-based system. Be sure to provide screenshots per sample conversation. Explain why you chose these conversations. Justify why the Chatbot is impressive on these situations.
- Give the worse conversations with your Chatbot. Be sure to provide screenshots per sample conversation. Explain why the poor performance of the Chatbot on these situations.
- Summarize the strengths and weaknesses

IV. Conclusions and Recommendations

Give a summary of what you have done, likely in 1-2 sentences. Enumerate the strengths and weaknesses of the Chatbot. Provide some recommendations to improve the Chatbot.

V. References (Follow the APA format)

VI. Appendix A: Interview Transcripts

The minutes should be signed by the interviewees to indicate their agreement with your understanding of their domain. However, if an interviewee does not wish to sign your minutes, provide the professor the interviewee's e-mail address and contact number so that the professor can get in touch with him/her.

VII. Appendix B: Contribution of Members

Name	Contributions

Notes:

- Algorithms found in the internet must be inspected and analyzed carefully. Deliberately using code from a source might result to a **failure in this project** if detailed explanation on how it works is not found in Chapter II of the document.
- Reusing previously submitted projects might also result to a **failure in this project.**
- Formatting requirements:
    - Include a Title Page.
    - Pages of the document must be numbered.
    - Check your document for spelling and grammar errors.
    - Proper usage of language and terms must be observed.

Submission Policy:

- Submit the .pdf documentation of your chatbot on **November 24**, 2359.
- Submission within 1 hour after the deadline (i.e. November 25, 0000 - 0100) will incur a 25% deduction on the grade.
- Submission more than 1 hour after the deadline (i.e. starting November 25, 0101) will not be accepted.
- The demo can be scheduled thru Canvas.
- **Plagiarized works will automatically be given a grade of 0.0 for the course.**

Format of the Title Page

<Title of your system>

A Chatbot Documentation
for the course on
Introduction to Intelligent Systems
(INTESYS)

Submitted by

<lastname, firstname> of group members
(in alphabetical order)

<Teacher's Name>
Teacher

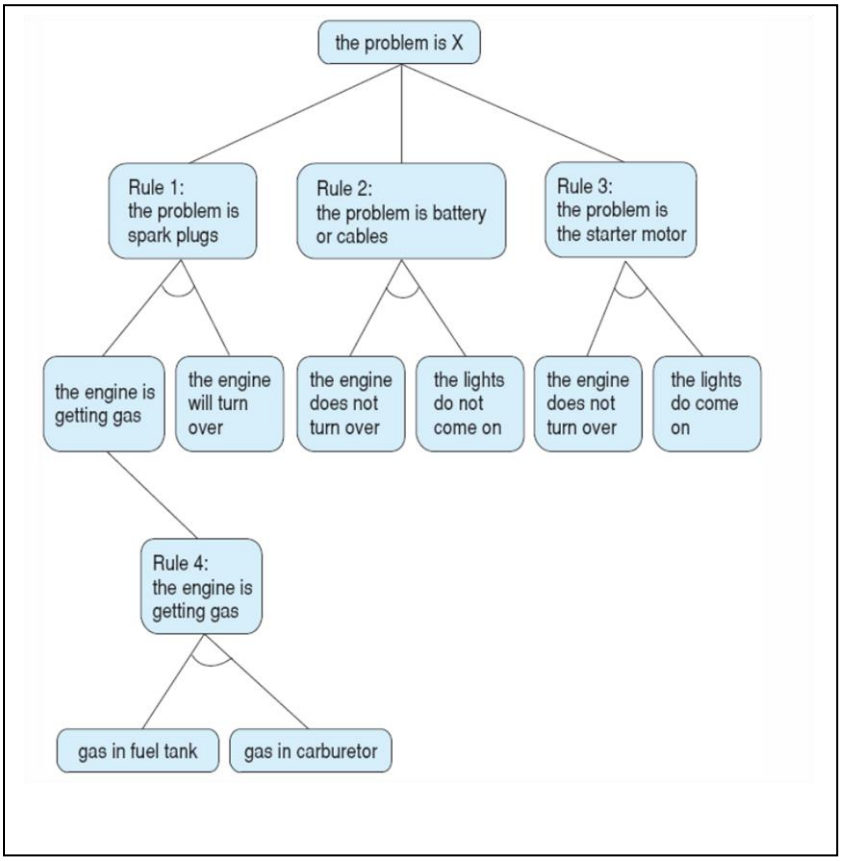
<Date of Submission>

A Backward Chaining Expert System Shell based on (Luger, 2005)
For further details, see (Luger, 2005), section 15.7.2.

```
% Expert system shell based on Luger
% To run, solve(fix(X),CF)

% solve(+,?)
solve(Goal,CF) :-
  print_instructions,
  retractall(known(_,_)),
  solve(Goal,CF,[],20).

print_instructions :-
  nl, write('You will be asked a series of queries.'),
  nl, write('Your response must be either:'),
  nl, write('a. A number between -100 and 100 representing'),
  nl, write('   your confidence in the truth of the query'),
  nl, write('b. why'),
  nl, write('c. how(X), where X is a goal'),nl.

% solve(+,?,+,+)
solve(Goal,CF,_,Threshold) :-
  known(Goal,CF),!,
  above_threshold(CF,Threshold).
solve(not(Goal),CF,Rules,Threshold) :- !,
  invert_threshold(Threshold,New_threshold),
  solve(Goal,CF_goal,Rules,New_threshold),
  negate_cf(CF_goal,CF).
solve((Goal1,Goal2),CF,Rules,Threshold) :- !,
  solve(Goal1,CF1,Rules,Threshold),
  above_threshold(CF1,Threshold),
  solve(Goal2,CF2,Rules,Threshold),
  above_threshold(CF2,Threshold),
  and_cf(CF1,CF2,CF).
solve(Goal,CF,Rules,Threshold) :-
  rule((Goal:-(Premise)),CF_rule),
  solve(Premise,CF_premise,[rule((Goal:-Premise),CF_rule)|Rules],Threshold),
  rule_cf(CF_rule,CF_premise,CF),
  above_threshold(CF,Threshold).
solve(Goal,CF,_,Threshold) :-
  rule(Goal,CF),
  above_threshold(CF,Threshold).
solve(Goal,CF,Rules,Threshold) :-
  askable(Goal),
  askuser(Goal,CF,Rules),!,
  assert(known(Goal,CF)),
  above_threshold(CF,Threshold).

above_threshold(CF,T) :- T>=0, CF>=T.
above_threshold(CF,T) :- T<0, CF=<T.

invert_threshold(Threshold,New_threshold) :-
  New_threshold is -1 * Threshold.

negate_cf(CF,Negated_CF) :-
  Negated_CF is -1 * CF.
and_cf(A,B,A) :- A =< B.
and_cf(A,B,B) :- B < A.
rule_cf(CF_rule,CF_premise,CF) :-
  CF is (CF_rule * CF_premise / 100).
```

```
% askuser(+,?,+)
askuser(Goal,CF,Rules) :-
  nl,write('Query : '),
  write(Goal), write(' ? '),
  read(Ans),
  respond(Ans,Goal,CF,Rules).

% respond(+,+,?,+)
respond(CF,_,CF,_) :-
  number(CF), CF=<100, CF>= -100. % no response issued because user enters a valid CF
respond(why,Goal,CF,[Rule|Rules]) :-
  nl, write_rule(Rule),
  askuser(Goal,CF,Rules).
respond(why,Goal,CF,[]) :-
  nl, write('Back to top of rule stack.'), nl,
  askuser(Goal,CF,[]).
respond(how(X),Goal,CF,Rules) :-
  build_proof(X,CF_X,Proof), !,
  nl, write('The goal '), write(X),
  write(' was concluded with certainty '), write(CF_X), write('.'), nl, nl,
  write('The proof of this is:'), nl,
  write_proof(Proof,0), nl,
  askuser(Goal,CF,Rules).
respond(how(X),Goal,CF,Rules) :-
  write('The truth of '), write(X), nl,
  write('is not yet known.'), nl,
  askuser(Goal,CF,Rules).
respond(_,Goal,CF,Rules):-
  write_rule('Unrecognized response.'), nl,
  askuser(Goal,CF,Rules).

% build_proof(+,?,?)
build_proof(Goal,CF,(Goal,CF:-given)) :-
  known(Goal,CF), !.
build_proof(not Goal,CF,not Proof) :- !,
  build_proof(Goal,CF_goal,Proof),
  negate_cf(CF_goal,CF).
build_proof((Goal1,Goal2),CF,(Proof1,Proof2)) :-
  build_proof(Goal1,CF1,Proof1),
  build_proof(Goal2,CF2,Proof2),
  and_cf(CF1,CF2,CF).
build_proof(Goal,CF,(Goal,CF:-Proof)) :-
  rule((Goal:-Premise),CF_rule),
  build_proof(Premise,CF_premise,Proof),
  rule_cf(CF_rule,CF_premise,CF).
build_proof(Goal,CF,(Goal,CF:-fact)) :-
  rule(Goal,CF).

write_rule(rule((Goal:-(Premise)),CF)) :-
  write('I am trying to prove the following rule:'), nl,
  write(Goal), write(':-'), nl,
  write_premise(Premise),
  write('CF = '), write(CF), nl.
write_rule(rule(Goal,CF)) :-
  write('I am trying to prove the following goal:'), nl,
  write(Goal),
  write('CF = '), write(CF), nl.

write_premise((Premise1,Premise2)) :- !,
  write_premise(Premise1),
  write_premise(Premise2).
write_premise(not Premise) :- !,
```

```
  write(' '), write(not), write(' '), write(Premise), nl.
write_premise(Premise) :- !,
  write(' '), write(Premise), nl.

% write_proof(+,+)
write_proof((Goal,CF:-given),Level) :-
  indent(Level), write(Goal), write(' CF='), write(CF),
  write(' was given by the user'), nl, !.
write_proof((Goal,CF:-fact),Level) :-
  indent(Level), write(Goal), write(' CF='), write(CF),
  write(' was a fact in the KB'), nl, !.
write_proof((Goal,CF:-Proof),Level) :-
  indent(Level), write(Goal), write(' CF='), write(CF), write(':-'), nl,
  New_level is Level + 1,
  write_proof(Proof,New_level), !.
write_proof(not Proof,Level) :-
  indent(Level), write((not)), nl,
  New_level is Level + 1,
  write_proof(Proof,New_level), !.
write_proof((Proof1,Proof2),Level) :-
  write_proof(Proof1,Level),
  write_proof(Proof2,Level), !.

indent(0).
indent(X) :-
  write(' '), X_new is X - 1, indent(X_new).

rule((fix(Advice):-(bad_component(X),fix(X,Advice))),100).

rule((bad_component(starter):-
  (bad_system(starter_system),lights(come_on))),50).
rule((bad_component(battery):-
  (bad_system(starter_system),not lights(come_on))),90).
rule((bad_component(timing):-
  (bad_system(ignition_system),not tuned_recently)),80).
rule((bad_component(plugs):-
  (bad_system(ignition_system),plugs(dirty))),90).
rule((bad_component(ignition_wires):-
  (bad_system(ignition_system),not plugs(dirty),tuned_recently)),80).

rule((bad_system(starter_system):-
  (not car_starts,not turns_over)),90).
rule((bad_system(ignition_system):-
  (not car_starts,turns_over,gas_in_carb)),80).
rule((bad_system(ignition_system):-
  (runs(rough),gas_in_carb)),80).
rule((bad_system(ignition_system):-
  (car_starts,runs(dies),gas_in_carb)),60).

rule(fix(starter,'replace starter'),100).
rule(fix(battery,'replace or recharge battery'),100).
rule(fix(timing,'get the timing adjusted'),100).
rule(fix(plugs,'replace spark plugs'),100).
rule(fix(ignition_wires,'check ignition wires'),100).

askable(car_starts).
askable(turns_over).
askable(lights(_)).
askable(runs(_)).
askable(gas_in_carb).
askable(tuned_recently).
askable(plugs(_)).
```


the problem is X
Rule 1:
the problem is
spark plugs
Rule 2:
the problem is battery
or cables
Rule 3:
the problem is
the starter motor
the engine is
getting gas
the engine
will turn
over
the engine
does not
turn over
the lights
do not
come on
the engine
does not
turn over
the lights
do come
on
Rule 4:
the engine is
getting gas
gas in fuel tank
gas in carburetor