CZ3005-Lab2: Chatting Bot Assignment-1: “Ten Questions”

TSR1, Cai Lingzhi, U1622184H

8/Nov/2018

1 Introduction

The Assignment-1 for Lab-2 designs and implements an answerer for the ten-question game base on the domain knowledge of the Olympic Game. The requirements given are:

- The program plays as an answerer;
- Maximum 10 queries can be asked;
- The queries are in the same form;
- There is one form of "decision" for the questioner;
- The program should be able to play the game for at least 5 time, with different result for each time.
- // According to the Professor, the specific topic may not necessarily be exactly finding the sports type as in the manual.

Instead of using the sample queries in the manual of guessing sports type, I decided to adopt a more interesting and interactive content. This project implements a ten-question game of guessing the **TOP 20 Olympic Athletes** that have the most medals. For every game, the program will randomly choose one among the twenty athletes. The user will have 10 chances to check relevant information, while the program will only return true or false for the enquiries. When the chances are exhausted, the program will automatically reveal the correct answer.

The data preparation for this project is done by python. All the remaining algorithms and implementations are completed with SWI Prolog.

In the following, Section 2 will provide detailed user guide for this program. Section 3 will explain the design and implementation of this program.

2 User Guide

Suppose that the working directory has set to the program's directory.

2.1 Load the program

```
?- ["lab2-ass1-athlete.pl"].
```

Execute the above code to load the program into the console.

2.2 Initialize the program

```
?- start.
```

Call the start method to initialize the program and clean up the buffer.

2.3 Query the program.

```
?- userCheck(<Item>, <Value>).
```

A query checks whether certain information about the athlete is correct.

For example,

```
?- userCheck(sport, swimming).
```

The above query checks whether the athlete is a swimmer.

Two lines of replies can be expected. The first line of output will remind the questioner the remaining chances, and the second line of output will return whether the query is true or false.

For example,

```
You have 5 chances remaining
true .
```

If the questioner has queried for more than 10 times, the program will always return false and show the correct answer as below:

```
The Question Count has Reached.
The Answer is: <CORRECT ANSWER>
false.
```

If the search process does not stop, press the ENTER after the true/false result has shown. Enter the new query or guess only after the "?-"symbol.

2.4 Make a Guess

The questioner can make a guess at any time (even after the number of queries exceeds 10), and the guess will not be counted.

The guess should be written as the following.

```
?- is(<YOUR GUESS>).
```

The program will return either **true** or **false** indicating that whether your guess is the same as the pre-determined answer.

2.5 A Small Cheat

If the questioner does not want to make guesses and wish to find the correct answer, the questioner can execute the below code at any time. The cheat query will not be counted.

```
?- is(X).
```

The program will return the correct answer in the following form:

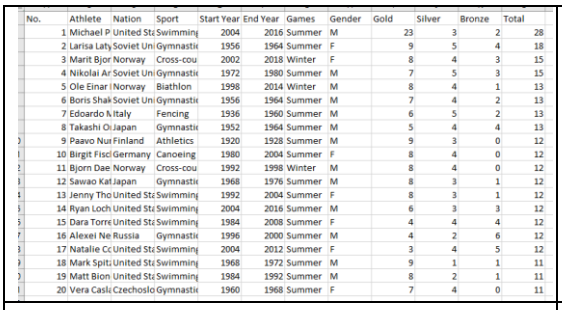
```
X = <CORRECT ANSWER>.
```

3 Implementation

3.1 Data Collection and Processing

All the primary data are retrieved from Wikipedia page "List of multiple Olympic medallists" on 5/Nov/2018 (Figure 1) (https://en.wikipedia.org/wiki/List_of_multiple_Olympic_medalists). The

primary data are initially stored in .csv file, and later written into Prolog Facts using Python (Figure 2). There are 220 lines of facts in total.



No.	Athlete	Nation	Sport	Start Year	End Year	Games	Gender	Gold	Silver	Bronze	Total
1	Michael P	United Sta	Swimming	2004	2016	Summer	M	23	3	2	28
2	Larisa Laty	Soviet Uni	Gymnastic	1956	1964	Summer	F	9	5	4	18
3	Marit Bjor	Norway	Cross-cou	2002	2018	Winter	F	8	4	3	15
4	Nikolai Ar	Soviet Uni	Gymnastic	1972	1980	Summer	M	7	5	3	15
5	Ole Einar	Norway	Biathlon	1998	2014	Winter	M	8	4	1	13
6	Boris Shak	Soviet Uni	Gymnastic	1956	1964	Summer	M	7	4	2	13
7	Edoardo M	Italy	Fencing	1936	1960	Summer	M	6	5	2	13
8	Takashi Oı	Japan	Gymnastic	1952	1964	Summer	M	5	4	4	13
9	Paavo Nur	Finland	Athletics	1920	1928	Summer	M	9	3	0	12
10	Birgit Fisc	Germany	Canoeing	1980	2004	Summer	F	8	4	0	12
11	Bjorn Dae	Norway	Cross-cou	1992	1998	Winter	M	8	4	0	12
12	Sawao Kat	Japan	Gymnastic	1968	1976	Summer	M	8	3	1	12
13	Jenny Tho	United Sta	Swimming	1992	2004	Summer	F	8	3	1	12
14	Ryan Loch	United Sta	Swimming	2004	2016	Summer	M	6	3	3	12
15	Dara Torre	United Sta	Swimming	1984	2008	Summer	F	4	4	4	12
16	Alexei Ne	Russia	Gymnastic	1996	2000	Summer	M	4	2	6	12
17	Natalie Co	United Sta	Swimming	2004	2012	Summer	F	3	4	5	12
18	Mark Spitz	United Sta	Swimming	1968	1972	Summer	M	9	1	1	11
19	Matt Bion	United Sta	Swimming	1984	1992	Summer	M	8	2	1	11
20	Vera Cask	Czechoslo	Gymnastic	1960	1968	Summer	F	7	4	0	11

Figure 1

```
% ========== 1. Initiate Data ========== %
rank(michael_phelps, 1).
rank(larisa_latynina, 2).
rank(marit_bjorgen, 3).
rank(nikolai_andrianov, 4).
rank(ole_einar_bjorndalen, 5).
rank(boris_shakhlin, 6).
rank(edoardo_mangiarotti, 7).
rank(takashi_ono, 8).
rank(paavo_nurmi, 9).
```

Figure 2

3.2 Control Logic

3.2.1 Initializing the Program

The program will be initialized using the **start** command. After the start command is called, the program will execute the following:

- Clean up the memory of the answer and counters in the previous round;
- Reset the counter to 10;
- Generate a random number and choose an athlete from the list accordingly;
- Record the athlete into the buffer;
- Print out the instructions for the questioner to proceed.

3.2.1.1 Memory Cleaning-up

The memory cleaning-up is completed using `retractall(<FACT>)` query. The `retractall(<FACT>)` query will retract the specified dynamic fact in the database. In this program, the answer is expressed in the fact of `is(<ANSWER>)`, and the counter records are stored in the fact of `counter(<NUMBER>)`, both of which are set as dynamic predicate. Therefore, we can clean up the memory by retract all of the `is()` and `counter()` fact, i.e.,

```
retractall(is(X)),
retractall(count(X)),
```

3.2.1.2 Reset the Counter

The counter can be reset by asserting a new fact `counter(<NUMBER>)`, i.e.,

```
assert(counter(10)),
```

3.2.1.3 Randomly Choose the New Answer

The randomly chosen answer is enabled by selecting random element in the name list, and is encapsulated into a Rule `choose()`. The program will first record the name of all athletes into a list, and then generate a random integer within the range of the length of that name list. Then the program will combine the random number with the list and returns the specific element as the new answer of this round of the game.

3.2.1.4 Record the New Answer

The new answer is then recorded into the buffer by adding an assertion of a fact `is(<ANSWER>)`. That is,

```
nameList(TheNameList),
choose(TheNameList, Athlete), % Randomly pick an athlete
assert(is(Athlete)), % Record the Athlete into the buffer
```

3.2.1.5 Provide Instructions

After all the set-ups have finished, the program will print out instructions for the questioner about what queries are expected. (Figure 4)

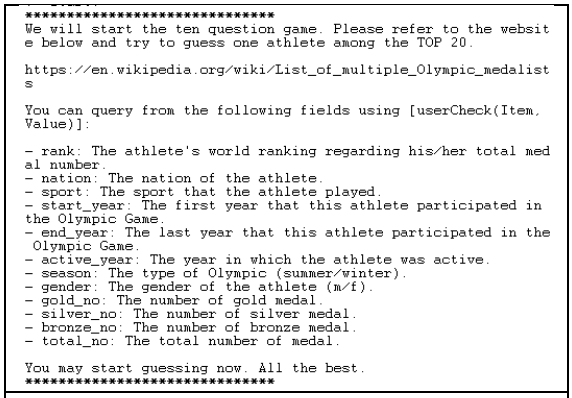


Figure 4

3.2.2 Checking the Information:

The questioner has ten chances to check the relevant information to guess the athlete by quiring `userCheck(Item, Value)`.

3.2.2.1 Allowed Input Field (Item)

The `userCheck(Item, Value)` can only accept `Item` in the following field:

`{rank, nation, sport, start_year, end_year, active_year, season, gender, gold_no, silver_no, bronze_no, total_no}`

The `Value` should specify the questioner's guesses. The program will return true/false as a reply.

3.2.2.2 Update the Counter

For each query received, the program will check and update the counter.

During the process of checking the counter, the program will search for the smallest count number recorded so far. If the smallest counter is less than zero, meaning that the chances for querying have been used up, the counter will return false, and print out the correct answer. Otherwise, the counter will decrease the count number, and update the latest count number by adding an assertion of it. After the update process, the counter will print out a reminder indicating the remaining chances.

3.2.2.3 Atomic Check

The specifically checking logic is defined using `check()`. After all the preparation has been done, and the program allows to reply to the query, the `userCheck()` will call the `check()` function. The check() function will query the knowledge base to see whether the statement is true.

For example, to check the sport that this athlete is playing, the `check()` function is implemented as:

```
check(sport, Sport):-
    is(Athlete),
    sport(Athlete, Sport).
```

This function will return **true** if the user's guess (Sport) is identical to the record in the knowledge base, i.e., the query `sport(Athlete, Sport)` returns **true**.

The program also implements a special check item called `active_year`. This is to check whether the athlete is playing given a specific year. The check function for this item will retrieve the `start_year` and `end_year` of the chosen athlete and determine whether the `active_year` is in between them. Checking the `between` relationship uses a recursion function that increments the lower bound of the range for one unit at a time, and then checks whether the target is the same as the lower bound. The function is implemented as the following:

```
between(Lo,Hi,N) :-
  integer(Lo),
  integer(Hi),
  between_sub(Lo,Hi,N).

between_sub(Lo,Hi,Lo) :- % unify the lower bound with the
                         % result
  Lo =< Hi.              % - if we haven't yet exceeded the
                         %   inclusive upper bound.

between_sub(Lo,Hi,N)  :- % otherwise...
  Lo < Hi ,             % - if the lower bound is less than the
                        %   inclusive upper bound
  L1 is Lo+1 ,          % - increment the lower bound
  between_sub(L1,Hi,N).   % - and recurse down.
```

4 Conclusion

The design and implementation is well satisfying the requirement for this assignment as the following:

- The program plays as an answerer:
    - The program will reply the queries using `userCheck()` and the confirm the guess using `is()`.
- Maximum 10 queries can be asked:

    - The program will keep track of the query number. It will remind the questioner when there is only one chance remaining, and will stop answering the queries after all chances have been used up.
- The queries are in the same form:
    - There is only one form of query: `userCheck()`
- There is one form of "decision" for the questioner:
    - There is only one form to make decision or guess: `is()`
- The program should be able to play the game for at least 5 time, with different result for each time:
    - The program will randomly choose a new answer and clean up the memory for every time it is initialized. Therefore, it will be able to play the game for infinite number of times.