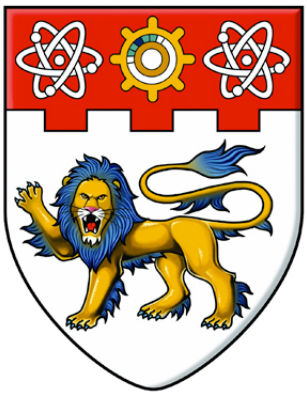

NANYANG TECHNOLOGICAL UNIVERSITY

CZ3005: Artificial Intelligence

Ten questions

Pontus Jarnemyr, N1902661D

Nanyang Technological University

2019-11-06

1 Introduction

This report presents a Prolog program called "Ten questions". The program accepts a number of queries from a player, and responds with either 'yes' or 'no'. Using a maximum of ten queries, the player is supposed to figure out which sport the program is thinking of. In order to improve readability, the queries will sometimes be referred to as questions in the report.

2 Facts in the program

The program needs to keep track of certain facts, such as the characteristics of each sport, which sports to choose from, and what sports have been guessed correctly before. These facts are presented in this section.

2.1 Static facts

Most facts remain static the whole game, such as the characteristics of all the sports. These facts are stored in a series of lists.

- Sports([...]) stores all the sports that the player can ask about.

```
1       sports([tennis,diving,football,...,  basketball]).
```

- Facts about each sport is stored in an individual list, such as tennis([...]), football([...]), etc.

```
1       tennis([court,score,ball,racket,...,teams_per_game=2]).
```

2.2 Dynamic facts

Some facts, like the number of questions the player has asked, need to change throughout the game. The program utilizes dynamic facts (modifiable facts) to keep track of these.

- Selected() keeps track of the current sport that the program is 'thinking of'.
- Already_played() keeps track of the sports that the player has previously guessed correctly.
- Counter() keeps track of how many questions the player has asked. A maximum of ten questions is allowed for each sport.

```
1       :- dynamic selected/1. :- dynamic selected/0.
2       :- dynamic already_played/1. :- dynamic already_played/0.
3       :- dynamic counter/1. :- dynamic counter/0.
4       selected(empty). already_played(). counter(0).
```

3 Game logic

The program's decision making is divided into a series of rules. This section will present some key rules and explain them in more detail.

3.1 game_over()

This one is fairly straight-forward; it ends the game. However, it can actually be considered a pair of 'nested' rules.
Game_over() is run whenever the game ends. It asks the player if they want to keep playing and reads their input. This input is passed to restart_game(Input). If the player entered 'y', play() is called. If they entered 'n', it aborts the program. If both failed, game_over runs again.

```
1  game_over() :-
2      print('Play again? (y/n)'),
3      read(Input),
4      restart_game(Input).
5  restart_game(X) :-
6      (X==y, play());
7      (X==n, abort);
8      game_over().
```

3.2 set_game_variables(List)

This is called upon from play(), when a new game is started. It tries to assert all the dynamic facts of the program, such as the selected sport and already_played sports. The argument List contains all the sports in the knowledge base.

Set_game_variables(List) starts by inserting all elements from already_played() into a list called Played. Played is then subtracted from the given List, resulting in a new list called Valid. Valid is a list that contains all the sports that the player has not guessed correctly.
If Valid is empty, the program terminates. If not, a random member from Valid will be placed into Sport. Sport is then asserted into already_played(), so it cannot be randomly selected next time. The previous value of selected() is retracted and replaced with the new Sport. Lastly, the counter is reset.

```
1  set_game_variables(List) :-
2      findnsols(100, X, already_played(X), Played),
3      subtract(List, Played, Valid), Valid\==[],
4      random_member(Sport, Valid),
5      assert(already_played(Sport)),
6      retractall(selected(_)), assertz(selected(Item)),
7      retractall(counter(_)), assertz(counter(0));
8
9      print('You guessed all the sports, well done!'), nl,
10     abort.
```

3.3 get_list(X, L)

Get_list (X, L) places all characteristics of a given sport X into the list L. It simply compares X to a number of strings and, when they are equal, places the corresponding list into L. The code has been somewhat redacted to save space.

```
get_list(X, L) :-
    (X==tennis, tennis(L));
    (X==diving, diving(L));
    ...,
    (X==basketball, basketball(L)).
```

3.4 check_counter()

Check_counter() ensures that the player is given only ten questions. It will load the value of counter() and check if it is larger than 10. If it is, the correct answer is printed and game_over() (section 3.1) is called.

```
check_counter() :-
    counter(X),
    (X<10);

    print('You ran out of guesses'), nl,
    print('The answer was '),
    selected(X), nl,
    print(X),
    game_over().
```

3.5 guess()

The program 'loops' around one main rule, called guess(). This is where most of the queries return when they have finished.
It starts by trying check_counter() (section 3.4). If successful, guess() reads the input from the player and performs the given query. If the player has run out of questions, game_over is called (section 3.1).

```
guess() :-
    check_counter(),
    read(Input),
    Input;

    game_over().
```

3.6 concat_all_facts(L)

This concatenates all the facts about each sport into a given list FullList. This is used whenever the player issues a query that needs all the available facts about each sport from the knowledge base.
Some code is redacted to conserve space, but the functionality can still be seen. The lists from all sports are placed into variables L1..L8. These lists are then appended to larger lists called Quart1List..Quart4List, which are in turn appended to even larger lists called Half1List and Half2List. Finally, the "half lists" are appended to the given list L.

```
concat_all_facts(L) :-
    tennis(L1),
    diving(L2),
    ...,
    basketball(L8),
    append(L1, L2, Quart1List),
    ...,
    append(L7, L8, Quart4List),
    append(Quart1List, Quart2List, Half1List),
    append(Quart3List, Quart4List, Half2List),
    append(Half1List, Half2List, L).
```

4 Player interaction

There are three queries available to the player: has(Guess), is(Guess), and all_options(). The program handles all queries in guess() (section 3.5).
This section describes how the program handles the queries in more detail. The main 'loop' of the program is shown in Figure 1.

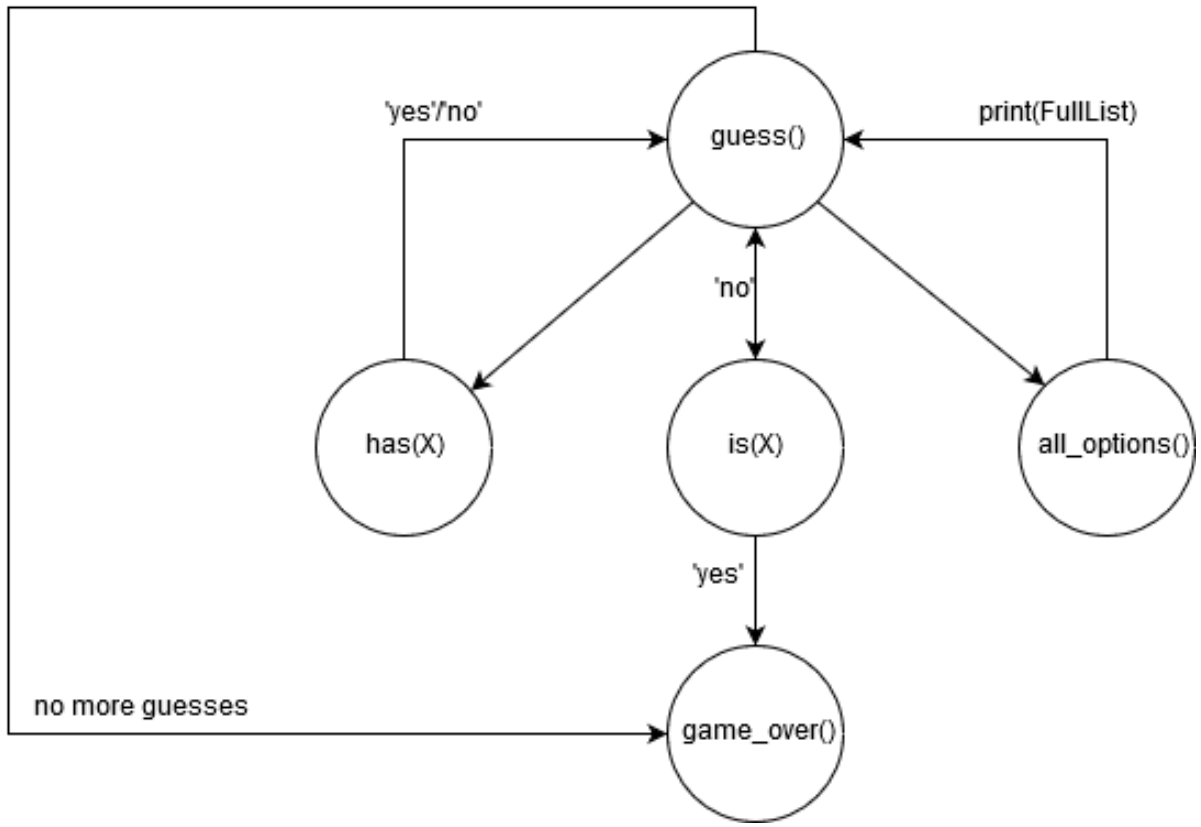

Figure 1: Each question returns to guess() at the end of their execution, except when the player guesses the correct sport in is(X).

4.1 play()

The game is first started by writing play() in the console window. This inserts all sports into list L and calls set_game_variables(L) (section 3.2). Finally, guess() (section 3.5) is called which lets the player start asking questions.

```
1 play() :-
2     sports(L),
3     set_game_variables(L),
4     print('Game started!'),nl,
5     guess().
```

4.2 has(X)

Has(Guess) is the query that answers whether the current sport has a certain characteristic or not. First, it loads the current sport into X by querying selected(X). It then calls get_list which will find the corresponding list to the current sport and store it in L. If the Guess is a member of L, the guess was correct. If the guess was not in L, then the program uses concat_all_facts(L) to check if X is a valid option for any of the sports. If it is not, the counter is not increased and the program returns to guess().

```
1  has(Guess) :-
2      selected(Sport),
3      get_list(Sport, L),
4      member(Guess, L),
5      print('Yes.'), nl,
6      increment(), /* This increments the counter by 1 */
7      guess();
8
9      concat_all_facts(FullList),
10     member(Guess, FullList), /* Check if Guess is valid */
11     print('No.'), nl,
12     increment(),
13     guess();
14
15     print('Invalid option'),nl,
16     guess().
```

4.3 is(Guess)

When the player wants to guess the answer, they will use the is(Guess) query. The program loads the current sport into variable X, and then compares Guess and X. If they are equal, the player was correct and the game is over. If they were wrong, the counter is increased by one and the program returns to guess().

```
1  is(Guess) :-
2      selected(X),   /* Load the current sport into X */
3      X==Guess,      /* Compare X to Guess */
4      print('Yes, it was '),
5      print(Guess), nl,
6      game_over();
7
8  /* If Guess is wrong: Print 'No' and increase counter */
9      print('No, it is not '),
10     print(Guess), nl,
11     increment(),
12     guess().
```

4.4 all_options()

The all_options() query is used to show the player all the things they can ask for in the has(X) query.
It by using concat_all_facts(L) (section 3.6). The list L then creates set S, which is free from duplicates. This set S is then printed, displaying all the possible options the player can use in the has(X) query (section 4.2), before returning to guess().

```
1 all_options() :-
2         concat_all_facts(L),     /* Get all options */
3         list_to_set(L, S),           /* Remove duplicates */
4         print('All possible options: '), nl,
5         print(S), nl,                                /* Print all
              options */
6         guess().
```

5 The program in action

This section shows how the program behaves and what it looks like during execution.

5.1 The full test run

Figure 2 shows the entire test game that was played. The player guesses four sports correctly before terminating the program. Since figure 2 shows the queries for the entire test run, the text is fairly small.
The player starts the game with play(). They ask for all the options. Given these options, they start asking about the sport. When they feel certain about the answer, they guess what sport the program is thinking of.

```
?- play().
|: all_options().
'All possible options: '
[court,score,ball,racket,referee,doubles,singles,net,serve,baseline,ballgirl,ballboy,grass,clay,teams_per_game=2,water,pool,judges,swimsuit,swimcap,
teamsize=1,teams_per_game=many,pitch,penalty,captain,manager,teamsize=11,timed,goggles,lane,shuttlecock,ring,gloves,knockout,course,sand,caddy,holes
,bag,fairway,clubs,basket,scoreline,teamsize=2]
|: has(ball).
'No.'
|: has(pool).
'No.'
|: has(doubles).
'Yes.'
|: has(shuttlecock).
'Yes.'
|: is(badminton).
'Yes, it was 'badminton
'Play again? (y/n)'|: y.
|: has(pool).
'Yes.'
|: has(judges).
'No.'
|: has(lane).
'Yes.'
|: is(swimming).
'Yes, it was 'swimming
'Play again? (y/n)'|: y.
|: has(ball).
'No.'
|: has(gloves).
'Yes.'
|: is(boxing).
'Yes, it was 'boxing
'Play again? (y/n)'|: y.
|: has(ball).
'Yes.'
|: has(grass).
'Yes.'
|: is(football).
'No, it is not 'football
|: is(golf).
'No, it is not 'golf
|: has(racket).
'Yes.'
|: is(tennis).
'Yes, it was 'tennis
'Play again? (y/n)'|: n.

% Execution Aborted
?-
```

Figure 2: Input and output of the full game.

5.2 Zoomed in test run

Figure 3 shows a zoomed-in game which shows the flow of events during a game. In this game, the player is lucky and gets the two first questions right. However, guessing the sport incorrectly twice causes them to ask a control question (has(racket)) before guessing the correct answer, which in this case was tennis.

This illustrates the cycle of guess() –> query(X) –> guess() outlined in figure 1; the program always returns to guess() after each query, except when the player guesses the sport correctly.

```
'Play again? (y/n)'|: y.
|: has(ball).
'Yes.'
|: has(grass).
'Yes.'
|: is(football).
'No, it is not 'football
|: is(golf).
'No, it is not 'golf
|: has(racket).
'Yes.'
|: is(tennis).
'Yes, it was 'tennis
'Play again? (y/n)'|: n.

% Execution Aborted
?-
```

Figure 3: Player asks questions to narrow down possible sports.

6 Conclusion

Some of the rules could probably be improved in order to easily add more sports into the knowledge base, such as all_options() and get_list(X, L). I did not manage to come up with a solution that allowed these rules to use less hard coded arithmetic. However, I am fairly pleased with how the program handles queries and how the flow of events can be narrowed down into a small graph (Figure 1).
Given more time, I would probably implement a GUI instead of issuing queries from the console window, and try to make the above-mentioned rules less hard coded.