Assignment 6

Due Friday, December 2nd, 11:59pm

You are encouraged to work with a partner on this assignment. I will also allow groups of three, but you must APPLY via email with your names, student numbers, and a description of your project.

Please post any requests for clarification on Matygo.

Deliverables:

Please submit all deliverables via the `handin` facility. Please include the following in your submission:

- Any necessary files to play your game. Be sure to clearly document and attribute all files. There will be penalties associated with poor documentation.
- Your documentation in your README file must include what is known as a "walkthrough" -- that is, a step-by-step description of how to completely solve the game as directly as possible.
- Also, for full credit, as part of your documentation you must identify at least two examples of responses your system gives that you consider undesirable or incorrect. No one's system will be perfect, so don't even try that argument!
- Your README file is also your chance to brag about what's especially cool. It is also acceptable to leave notes to your marker **in** your game, too.
- Your game must start with the predicate, play/0.

Goals:

The purpose of this project is to demonstrate the power of logic programming via Prolog, as well as Prolog's built-in grammar support for processing natural language (NL). You will undoubtedly recognize some of Prolog's limitations, as well.

In addition to the overarching goals above, we have the following learning goals:

1. Use the DCG framework in Prolog to write grammars
2. Gain experience designing and implementing a larger Prolog program
3. Explore efficiency in Prolog (e.g., goal/rule ordering and cuts).
4. Encounter several common idioms of Prolog programming (like the "repeat" looping predicate).

The Assignment:

In this project, you will design a short, text-based adventure game. In its entirety the game should take about 5-15 minutes to play through to completion. Given the goals above, you are free to be as creative as you wish!

Your grade will reflect a number of things:

1. Evidence of effort
2. Original thought -- BE CREATIVE!
3. Evidence of planning
4. Quality of code (clean style, no redundancy, proper use of recursion, efficiency)
5. Overall complexity
6. Quality of final product

Project Resources

You may need the following resources during this project (available as part of this Matygo resource collection):

1. A **tokenizer** (from Covington, 1994).
2. The ProNTo set of Prolog tools.
3. The **command loop interpreter** and the **get command code** provided in this document.
4. You are welcome to use other resources, within reason, however, you must be sure to give credit in each case. Use your judgment. If a text adventure game in Prolog is out there, trust me, I've seen it. Take that as a challenge to do your own work, not to outsmart me.

Suggested First Tasks

Your first goal should be to read and understand the project requirements. From there you can identify the major tasks, and subtasks. Now is a good time to talk about division of labour, and expectations (be very clear on these with one another... it's better you risk being pedantic now, than risk a mis-understanding later).

If you have never played a text-based adventure game before and are unclear, the wikipedia has a decent entry on them. During this first week you should learn about what they are, what is needed for them, and put together the outline for your game: what is the objective, how do you achieve that objective, et cetera.

It's up to you to devise a grammar that will allow you to read in commands or "sentences". Do note, however, that your sentence will already be a list of terms by the time your grammar needs to deal with it (the tokenizer (same as above) will take care of this -- also, see the code provided for you below). Therefore, familiarize yourself with what this will entail (i.e. learn how to use the tokenizer, and how to build a grammar... we'll be covering the latter part in class).

You don't have a tonne of time, so get started ASAP, and delegate tasks with your partner.

Main Project

This section lists a set of requirements your final adventure game must meet. You need not work on these tasks in the order listed. Many of the tasks do not directly depend on each other, and even where they interact to an extent, much progress can be made independently.

Feel free to creatively interpret these requirements. Think of the spirit of the requirements (and ask if you're unsure).

1. Create a **knowledge base** that describes the world of your game. This does not need to be overly complicated-- **better is to start with a simpler world and add to it later if you have time**. In your writeup describe your world, and the rules you used to describe it. If there is consistency among the rules, you do not need to describe every single one (if they are self-explanatory).
2. Your game must make use of an **inventory** system whereby the player can pick up items from the world and keep them in an inventory. This inventory can be called up from the prompt at any time. There must be at least 5 items in the world that can be picked up. You should also be able to "drop" items-- the system must keep track of where they are dropped.
3. There must exist an **overall objective** to the game. For example, you might have to find an item, or accomplish a simple task. Having a theme will make your task more fun.
4. You must be able to **interact** with things in your world, as well as keep track of the **state** of these things. E.g. A call to "open a door" means that that door remains open until it is closed. A door cannot be both open and closed.
5. You must have at least one **"puzzle"** in your game. For our purposes, a puzzle will entail any task that cannot be completed until all the conditions are met. For example, "turn on the TV" might be completed by a predicate that is only true (satisfiable) when the TV is plugged in, the power is on, there are batteries in the remote, and the remote is facing the TV.
6. You must have a **look command** available to the user that describes where they are, and where they can go from there. For example:
```
> look
You can see the following things:
apple
table
broccoli
You can go to the following rooms:
cellar
office
dining room
```
7. You must make use of a **DCG-based grammar**. It does not need to be a thorough English grammar-- just enough to handle basic commands within the confines of the game. You will not reasonably be able to anticipate all possible variations of a command issued (e.g. "open door", "open the door", "please open the door", etc). Thus, you must be clear in the instructions to your user the sorts of structures you allow. As an added challenge, you can explore ways for incorporating more flexibility in your interface.

You will need a predicate **play/0** that starts an interpreter loop. The interpreter loop should accept the game commands, as well as "help", and "quit" (you can add an optional "hint", if you like), from the user, allowing them to navigate within your world. Here's an example of some Prolog code driving such a loop (you can use this if you like):

```
command_loop:-
        repeat,
        get_command(X),
        do(X),
        % If end_condition is true, then X is "quit"
        (end_condition; X==quit).
```

Here's an example session with a sample game:

```
> go to office
You are in Kim’s office.

You can (barely) see the following things:
desk
piles and piles o’ paper
You can go to the following rooms:
hall
> get desk
I don't understand, try again or type help.

> pick up desk
You can't pick up a desk!

> go to hall

You can see the following things:
huge line up of students
axe murderer
You can go to the following rooms:
office
room 238

> pick up axe murderer
You now have the axe murderer! This might come in handy.

> use axe murderer on room 238
You deposit the now annoyed axe murderer in room 238.
Room 238 is now off limits.

> go to room 238
No!  There’s an annoyed axe murderer in there.
```

et cetera...

Some Extra Help

To help you out, I will provide the code necessary for translating a user command as typed in at the prompt, to a Prolog structure (i.e. predicate). You can also use the command loop code from above.

Feel free to modify this code, or expand it:

```
get_command(C):-
     readlist(L),
     command(X,L,[]),
     C =.. X,!. % Note the new predicate here (see
                % discussion immediately following)
```

The `univ` predicate

The line, `C=.. X,!.`, above, introduces a new built-in predicate called `univ`, written as the infix operator `=..` (equals followed by two periods). This predicate translates a prediate and its argument into a list whose first element is the predicate name and whose remaining elements are the arguments. It also works in reverse, building a predicate out of a list of elements. Thus, if we read in a command sentence as a list of tokens: `[open,door]`, `univ` allows us to generate the following: `open(door)`. This is done like so: `X=..[open,door]` where `X` will contain the desired predicate, namely `open(door)`.

In the example command loop above, we could then pass in `open(door)` to `do/1` to "execute" the command. (So awesome!)