Instituto Tecnológico y de Estudios Superiores de Monterrey

Campus Querétaro

Curse

The Prolog Text Adventure

Documentation

Programming Languages
August – December 2020

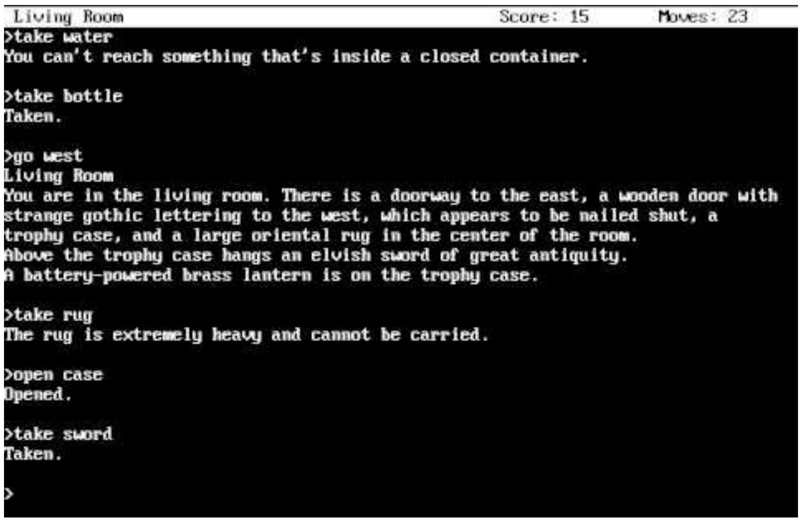
Eduardo González Melgoza
A01701446

Index

What Is A Text Adventure?

Text adventures, also referred to as interactive fictions, date back to the year 1976. They are a form of software in which the user is presented with a text-based description of various environments, situations and/or characters. These elements can be interacted with and influenced using simple text commands. Interactive fictions are also viewed as a form of video games, usually belonging to the adventure games or role-playing game (RPGs) genres thanks to the nature of their functionality. In text adventure games, the player normally takes on the role of a main character in the narrative, interacting by first reading what the program describes, and then inputting a command congruent to what is outputted onto the screen.

Even though these games lack a use of a graphic interface, there is a physical dimension in which the game takes place. In-game descriptions include imagery of the player's surroundings, such as the current room, interactive items and other non-player characters (NPCs). The player can input commands such as "go north" or "take object" to move from one room to another or interact with the characters and objects in each room, prompting the program to give a feedback description of how the world changes as the player affects the environment. Some notable examples of famous text adventures include Colossal Cave Adventure (1976), the Zork series (1980, 1981, 1982) and Cypher. Figure 1 shows a screenshot taken from an online version of Zork I[1]. In it, the gameplay can be appreciated, the player's input is shown after the '>' and the game feedback and descriptions follow each command given.



```
 Living Room                                          Score: 15        Moves: 23
>take water
You can't reach something that's inside a closed container.

>take bottle
Taken.

>go west
Living Room
You are in the living room. There is a doorway to the east, a wooden door with
strange gothic lettering to the west, which appears to be nailed shut, a
trophy case, and a large oriental rug in the center of the room.
Above the trophy case hangs an elvish sword of great antiquity.
A battery-powered brass lantern is on the trophy case.

>take rug
The rug is extremely heavy and cannot be carried.

>open case
Opened.

>take sword
Taken.

>
```

Figure 1

[1] https://classicreload.com/zork-i.html

Objective

The objective of this project is to apply and expand on the acquired knowledge of the logic programming paradigm to design and code a functional text adventure. The game will be coded using the Prolog programming language.

Result

The project resulted in the text adventure game named Curse (the source file's name is curse.pl). Curse is a fantasy adventure RPG text adventure game with light horror elements. It was coded using the Prolog programming language and the logic programming paradigm. The player can advance through the game's plot by inputting some of the classic commands also present in other famous text adventures like the "look." command to look about the player's surroundings or "i." to check their inventory. The game has a concrete plot an objective and an ending.

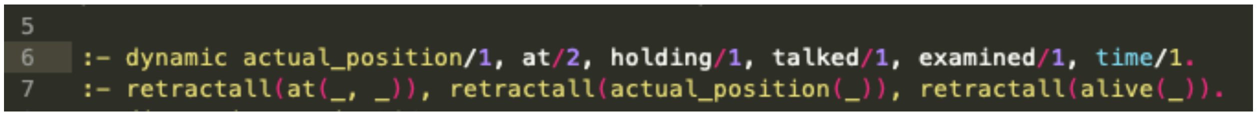
Functionality

Since Curse was programmed in Prolog, the functionality of the game is based on a series of rules and facts, as well as unification of different values to various variables. To understand the basic concepts of Prolog, a tutorial for beginners available in the following link: http://www.doc.gold.ac.uk/~mas02gw/prolog_tutorial/prologpages/. This tutorial will explain some central concepts like facts, arguments, unification and recursion, as well as interactive examples that will help get a better understanding of them.

In the source code of Curse, there are some techniques and predicate properties being used that are not explained in the aforementioned tutorial. These techniques are as follows:

dynamic/1

`dynamic/1` is a predicate property that indicates that the definition of the indicated predicate may change during execution. To cycle between the various possible definitions of a dynamic predicate, the use of the `assert/1` and `retract/1` terms is needed. The use of `dynamic/1` can be observed in line number six of the curse.pl source file.



```
5
6  :- dynamic actual_position/1, at/2, holding/1, talked/1, examined/1, time/1.
7  :- retractall(at(_, _)), retractall(actual_position(_)), retractall(alive(_)).
```

Figure 2

Figure 2 shows the sixth line of Curse's source code. The predicate property `dynamic/1` is being used to modify the predicates `actual_position/1`, `at/2`, `holding/1`, `talked/1`, `examined/1` and `time/1`.

assert/1

`assert/1` is a meta-predicate in Prolog that adds the argument it receives to the Prolog knowledge base. It is useful when a new rule or fact needs to be added to the knowledge base during the execution of a Prolog program. It is important to note that it can only be applied to dynamic predicates. It has two variants: `asserta/1` and `assertz/1`. The former pushes its argument to the beginning of the knowledge base, while the latter adds it at the end. Though deprecated, `assert/1` functions the same way as `assertz/1`. A use case example for `assert/1` is shown below:

```
?- assert(rich(mary)).
true.

?- rich(mary).
true.

?- assert((happy(X) :-
      rich(X),
      healthy(X))).

?- assert(healthy(mary)).
true.

?- happy(X).
X = mary
```

retract/1

`retract/1` is a meta-predicate in Prolog that removes the argument it receives from the Prolog knowledge base. It works in conjunction with `assert/1`, making it important to note that it can only be applied to dynamic predicates as well. A use case example for `retract/1` is shown below:

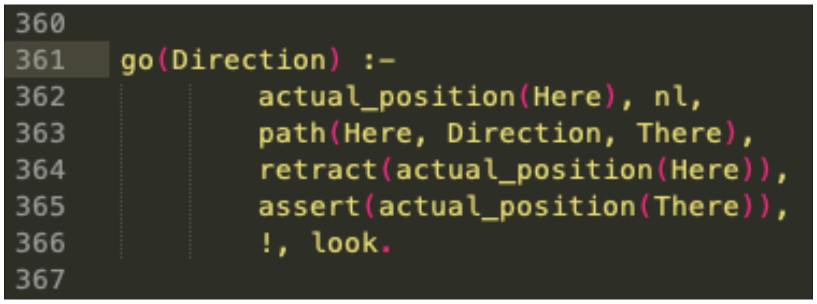
```
?- assert(likes(mary, pizza)).
true.

?- likes(mary, pizza).
true.

?- retract(likes(mary, pizza)).
true.

?- likes(mary, pizza).
false.
```

`assert/1` and `retract/1` are both used various time throughout the source file of Curse. Figure 3 shows an example of a situation in which they are used.



```
go(Direction) :-
        actual_position(Here), nl,
        path(Here, Direction, There),
        retract(actual_position(Here)),
        assert(actual_position(There)),
        !, look.
```

Figure 3

Here, `assert/1` and `retract/1` work together to change the players current position, retracting is previous position fact from the knowledge base and asserting the new one.

retractall/1

`retractall/1` functions in the same manner as `retract/1`, except it eliminates from the knowledge base all of the predicates that match its argument. Figure 4 shows the usage of `retractall/1` in the source code of Curse. In this instance, it is being used to clean the knowledge base before assigning actual definitions to some of the dynamic predicates.

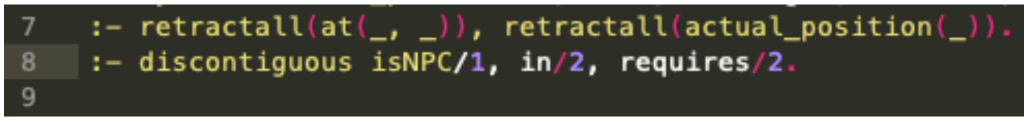
discontiguous/1

`discontiguous/1` is a meta-predicate that indicates that the defining clauses of the predicate that it receives as argument, are not restricted to be consecutive, whereas they can appear anywhere in the source file. This is an example of how to use `discontiguous/1`:

```
:- discontiguous dog/1.

dog(spike).
cat(mittens).
cat(tabby).
dog(spot).
```

In this example, the different predicates for dog are not defined one after the other. Without the first line of the example, this would output a warning. In the source file for Curse, `discontiguous/1` is used as follows:



```
:- retractall(at(_, _)), retractall(actual_position(_)).
:- discontiguous isNPC/1, in/2, requires/2.
```

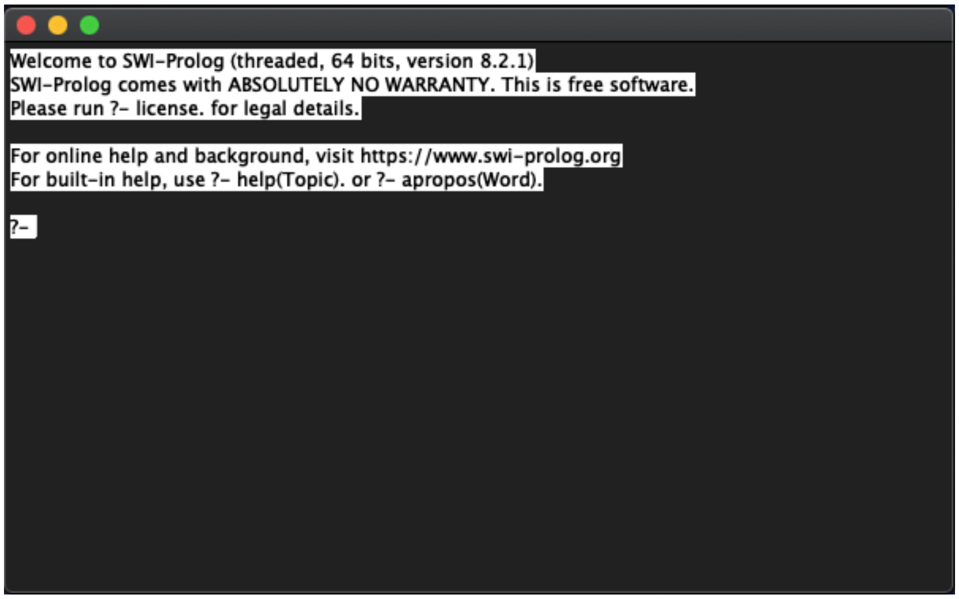
Figure 4

Figure 4 shows how `discontiguous/1` is being applied to the `isNPC/1`, `in/1` and `requires/1` predicates.

Setup

To be able to play Curse, it is recommended to run the code using the SWI-Prolog IDE. This is available to be downloaded from the following link: https://www.swi-prolog.org/download/stable. Please note that there exists an online version of SWI-Prolog, however, this version does not allow for the adequate functionality to work properly[2]. Once SWI-Prolog has been installed, running it should open a window like the one shown in figure 5 (screenshot taken on macOS).

[2] The game works by asserting and retracting the different predicate definitions, and this does not work properly in the online IDE.



```
Welcome to SWI-Prolog (threaded, 64 bits, version 8.2.1)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

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

?-
```

Figure 5

Due to the nature of the Prolog language, the game's code is a series of predicates that have to be consulted and queried in order to play through it. To load the game's source code file into SWI-Prolog and start playing, run the following command, using the absolute path of where the source file is saved:

```
?- consult('<file/absolute/path.pl>').
```

Example:

```
?- consult('/Users/lalogonzalez/repos/Curse/curse.pl').
```

Walkthrough (and all possible paths)

To begin to play Curse, the aforementioned setup must be completed prior to beginning. Once the setup is done, enter the following command to start the game:

```
?- start.
```

After the command has been executed (which in reality is a query), the game controls will be explained, followed by the initial game scenario description.

Bibliography

Lu, J. (N. D.) Prolog A Tutorial Introduction. Computer Science Department. Bucknell University. Accessed on: October 2020. Available at: http://academicos.azc.uam.mx/cbr/Cursos/UEA_12p_Log/PrologIntro_eng_good.pdf

Burnett, I. (2015) The Text Adventure: Relic of Gaming History, or Timeless Medium? The Artifice. Accessed on: October 2020. Available at: https://the-artifice.com/text-adventure-gaming-history/

Ireson-Paine, J. (2020) Why Use Prolog? Joselyn Ireson-Paine Consultancy. Accessed on: October 2020. Available at: http://www.j-paine.org/why_prolog.html

N. A. (2020) Predicate consult/1. Prolog Documentation. Accessed on: October 2020. Available at: https://www.swi-prolog.org/pldoc/man?predicate=consult/1

N. A. (2020) Predicate dynamic/1. Prolog Documentation. Accessed on: October 2020. Available at: https://www.swi-prolog.org/pldoc/man?predicate=dynamic/1

N. A. (N. D.) Zork I. Classic Reload. Accessed on: October 2020. Available at: https://classicreload.com/zork-i.html