Programming Languages

Final Project

Prof. Benjamin Valdés

Eric Fernando Torres Rodríguez

A01700249

22/11/2021

Introduction: -

Text base games trace as far back as teleprinters in the 1960s, when they were installed on early mainframe computers as an input-and-output form. These games shaped the interaction between users and the fictional environment that surrounded them.

The core functionality of these games consists in the input and output of data in form of text.

Solution: -

Based on my passion for videogames in all their presentations and as a tribute for the father of the now games that I enjoy, I created a text base video game using Logic Programming with Prolog, which interact with the user in an interactive word with items, puzzles, and enemies, all of these using prolog language commands.

The rules of this game are base in several predefine actions that as a user you can implement to interact with your environment and surroundings, each action can have or not consequences, lead to another place or even to death.

Prolog:

Since text base games consist in simple instructions and in a finite number of results were most cases end up asking if something can be done or not, therefore prolog is an easy approach to develop a solution to an interactive and immersive experience because prolog is a logic language that is particularly suited to programs that involve symbolic or non-numeric computation.

Prolog consists of a series of rules and facts. A program is run by presenting some query and seeing if this can be proved against these known rules and facts.

Prolog is a language also uses two main functionalities that make them a great program for a text-base game: unification and backtracking, with the use of these we can unify facts in a knowledge base and create predicates and rules that can use the backtracking to create situation where if some criteria is not accomplish, we can have another result. For example, if a rule were a certain item is needed is accomplish, we can have access to a specific area of the game or not.

Implementation: -

1. Simple facts

Prolog facts make a statement over something, in the next figure we ca see for example that we declare that the first position of the user is the monastery or that the cult is alive.

```
/*Start in the spawn*/
position(monastery).

/* This fact specifies that the cult is alive. */

alive(cult).
```

2. Fact with arguments to create connections

One of the core functionalities of out text-base program are the connection between locations in our world, we use facts with arguments to create the imaginary roads that take the user to different locations. As an example, in the next image we can see that we initialize a “path” that declare that form the monastery you can go to the south and arrive to the forest.

```
/* Monastery*/
path(monastery,south,forest).
path(monastery,west,river).
path(monastery,east,town).
```

3. Rules and backtracking

Another pilar of the program and maybe the one most important of all, are the rules, which allow us to make conditional statements about our world. Each rule can have several variations, called clauses. These clauses give us different choices about how to perform inference about our world.

In the next figure we can see a clear of example of the use of rules and backtracking, first we declare a rule that tell us that the path from the cave_entrance going to the east to arrive to the cave will be true only if the fact item_on(lantern, in_hand) is also true. Consequently, the second rule that only instructs to write a message and then call the function die will occur.

```
/* Cave entrance*/
path(cave_entrance,east,cave):-
  item_on(lantern, in_hand).

path(cave_entrance,east,cave):-
  write('It is to dark to move foward'), nl,
      !, fail.

path(cave_entrance,west,forest).
```

4. Unification

In order to know retrieve something from our knol