



**Tecnológico
de Monterrey**

Programming Languages

Prof. Benjamin Valdés

Martin Noboa – A01704052

August – December 2021

November 18th, 2021

Contents

Why this project?	2
Origin of text-based games	2
Success of text-based games	2
Solution	3
Why Prolog?	3
Explanation of code	3
Game	7
Narrative	7
Solution	7
Results	9
Evidence	9
Conclusions	9
References	11

Why this project?

Origin of text-based games

What is a text-based game? Dating back to the 1960s, text-based games are a form of electronic gaming where the user interacts precisely with that, a text-based interface, this is opposing a more “traditional” electronic game, based on vector or graphic interface. The interactions between the game and the user are all in ASCII characters, such as words.

Since they date back to the 1960s, when the most basic input and output computers were starting out. Back then, video terminals were in early development and very expensive. The hardware limitations made the games simple, without many of the to-go features you expect of a game nowadays. For example, since gaming sessions were brief, there was no saving feature.

There are several genres of text-based games, but the most predominant are adventure text-based games. They are the most known games and are often represented in modern media, such as [this](#) example in The Big Bang Theory (actually, this episode was partly the reason for this project selection).

Success of text-based games

The development of this type of game started because it was all that could be made with the available tools at the time. Like most games of the era, imagination was a powerful motivator to keep on playing this genre of games. However, they grew past their era and remain as one of the most iconic types of games in the history of gaming. When one thinks on the staples of gaming, aside from Pacman, Galactica and Mario, text-based adventure games are right up there with them.

A growing community still supports this game, maybe because of nostalgia or simply because of the different mechanics it represents when compared to modern day games.

This all led me to choose this particular project, to develop a text-based game in Prolog

Solution

Why Prolog?

Prolog is a logic programming language, and it is intended to work as a declarative language. Because of the way Prolog works, using facts/predicates and rules, it is a very “straightforward” language, this is given the paradigm that employs. So, what makes Prolog the correct tool to accomplish this project? Since the objective is to create a text-based game, the knowledge base and relations allow us to manipulate the predicates and create rules to create the right gaming experience. Don’t be mistaken, this does not mean that Prolog or this programming paradigm are the only way to approach this problem, but since Prolog works with facts and the relationships between these facts, it certainly makes this solution simpler to implement.

As I mentioned before, Prolog works with the combination of the knowledge base and rules that work with said knowledge base. The knowledge base is a group of *facts* that are the base of the program. In fact, as mentioned by Dennis Merritt, *“the term program does not accurately describe a Prolog collection of executable facts, rules and logical relationships”* (Merritt, 2016). For the sake of clarity however, we will continue to refer to it as a program.

In the next few sections I am going to explain each of the parts that compose the code, as well as explain some of the techniques used in this project.

Explanation of code

Knowledge base

A knowledge base in Prolog is a “database of everything that is known true to the running Prolog program” (IBM, n.d.). Once the program is compiled, these assertions *can not* be changed, unless they are declared as dynamic. We will go into detail about dynamic predicates later, but for now let’s focus on the basic predicates that make up the knowledge base.

```
% Describe locations
location(road).
location(forest).
location('deep forest').
location(cave).
location('deep cave').
location(swamp).
location(stable).
location(castle).
location(staircase).
location(basement).
location('wine cellar').
location('second floor hall').
location(hallway).
location(library).
location('living room').
```

Image 2. Predicates

In the *Image 1*, we can observe the declaration of the locations available in the game. This is the most basic type of predicate, a simple declaration. Once we declare the locations, these locations are **true**. After this we can begin to play around with what statements we wish to be declared as **true** before the program compiles. As seen in *Image 2*, now we declare a *relationship* between 2 locations. In this way, when we are working with logic, can I go from point A to point B? If the predicate exists, then it will be true, else it will be false.

```
% Describe connections between locations
connection(road,forest).
connection(forest, 'deep forest').
connection('deep forest',swamp).
connection('deep forest',cave).
connection(cave, 'deep cave').
connection(swamp,stable).
connection('deep forest',castle).
connection(stable,castle).
connection(castle, 'castle hall').
connection('castle hall',staircase).
connection(staircase, basement).
connection(staircase,'second floor hall').
connection('second floor hall', hallway).
connection(basement,'wine cellar').
connection(hallway,library).
connection(hallway,'living room').
```

Image 2. Relationship between predicates

Rules

Moving on from predicates, we arrive at another part of the knowledge base: the rules. Once we have the predicates we will be working with, we can begin to create rules around these predicates. In this section we begin to implement logic. There are a lot of people that translate rules from logical programming paradigm to functions in functional programming paradigm, while it seems like a good comparison, it hurts more in the long run than it helps. Rules have a specific aspect proper of the logical programming paradigm, which is the working context. Since we know predicates can not change their assigned value once the program compiles, what about variables? They also hold their value only on the context they are declared. You can think of it as a for loop in functional programming, as seen in *Image 3*.

```
for(int i = 0; i < 10; i++){  
    //  
}
```

Image 3. For loop example

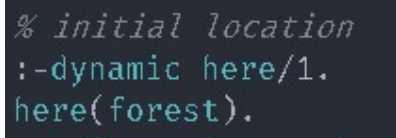
Much like how the variable *i* exist only in the context of the for loop, variables only exist on the context of the rule. So now that we know this, in the *Image 4*, we can observe an example of a rule in the project. If you go back to *Image 2*, you'll notice that the relationships are one way, since the order of the predicates matter, i.e., is not the same to write **location(a,b)** as **location(b,a)**. So, the following rule take the location predicate and makes it go both ways.

```
% Rules for making connection reciprocate  
connect(X,Y):-  
    connection(X,Y).  
connect(X,Y):-  
    connection(Y,X).
```

Image 4. Rule example

Dynamic Predicates

Finally, we get to what I consider the most important section of the project, or at least what I consider so. We began working on the premises that variables only work in their context and predicate's values cannot change once the Prolog file compiles. So, how would we keep track of say our inventory or our location? We can do this through *dynamic predicates*. Dynamic predicates allow us to manipulate the database as the program runs. Full disclosure, it is not best practice for most cases, however, it is necessary for our program. To work with dynamic predicates, we must explicitly declare them as seen in *Image 5*.



```
% initial location
:-dynamic here/1.
here(forest).
```

Image 5. Dynamic predicate declaration

Once a predicate is declared as dynamic, it can be manipulated through Prolog commands, such as:

- **Retract** – when an atom (basic data type in Prolog) is unified to a predicate, retract will remove it without interfering with the logic clause.
- **Assertz** – “inserts” a predicate into the database.

The combination of these 2 commands is very powerful, and very useful in the cases that apply in our project.

Game

Narrative

Now, we will discuss a little about the game itself. The idea for the game came at a time where I was watching both Supernatural and the Big Bang Theory, so I took some elements from each. The game's name is "Forgotten", the setting is in a forest with a nearby abandoned castle. To beat the game, the player must get back his voice recorder to remember who the player is and what was the purpose of the visit to the location. There are some puzzles the player must beat to finish the game, like collect some items and get the key to a certain door. In the *Image 6*, we can see a draft of the game map at the early stages of development. Next, on *Image 7*, we can see a digitalized version of that same map which is implemented in the final version.

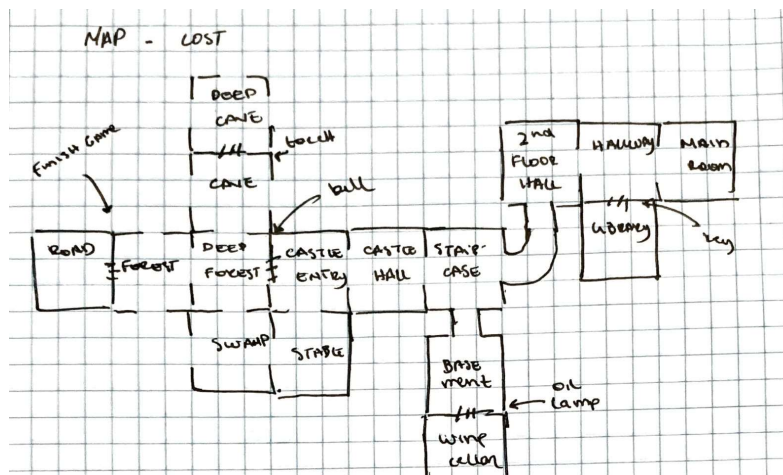


Image 6. Map draft

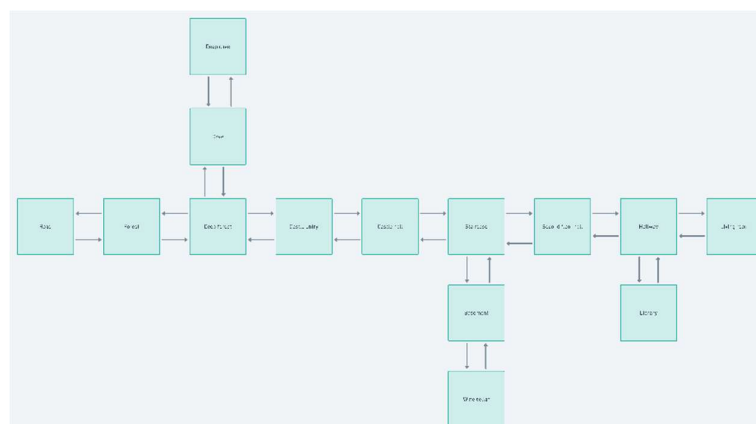


Image 7. Map digital ver

Solution

Finally, just in case you need it and do not wish to play the game yourself, here is the solution to the game.

1. Once you get to the living room, an old man holding your recorder will yell that he needs his wine and his book.
2. Go to the castle hall to get an oil lamp.
3. Head to the basement, there you will get the key.
4. In the wine cellar you can get the wine.
5. With the key, you can now enter the library.
6. In the library you take a book.
7. Once you have the items, you can leave them on the table of the living room and get your recorder.

Results

Evidence

```
PS C:\Users\Martin Noboa\Desktop\Languages\ProgrammingLanguages-FinalProject> swipl --quiet AdventureGame.pl
1 ?- info.
You are in the forest.
The available things are:
You can move to:
road
deep forest
true.
```

```
2 ?- go('deep forest').
You moved from forest.
You are in the deep forest.
The available things are:
branch
You can move to:
forest
swamp
cave
castle
```

```
2 ?- take(branch),go('castle').
Taken branch into inventory.
You moved from deep forest.
You are in the castle.
The available things are:
lamp
You can move to:
deep forest
stable
castle hall
```

Setup instructions

To keep everything in order and easily accessible, the setup instructions are in the GitHub repository's README file. You can access the repository following [this link](#).

Conclusions

The challenge presented by this project was great. Although I consider that logical programming and Prolog being my “strong suit” this semester, there was quite the learning curve for this project. Aside from remembering the class topics, there were some others that were not covered and required investigation on my end. I really enjoyed the process of learning how a text-based game works and its history as well.

The areas of opportunity for this project as many, starting by the extension of the game. There is so much more I would have liked to implement and other mechanics I would’ve liked to try, however I am still very pleased with the result. I believe that the extension of the project was measured correctly, to deliver a complete project without compromising its complexity. Still, it is a project that I will surely revisit later to improve on it.

I am always thankful to have the opportunity to apply what was learned in a practical project, so this project was fun for me. While I chose to work on a text-based game this time around, I know Prolog has some artificial intelligence applications so that is something I want to explore as well.

References

Bibliography

Blackburn, P., Bos, J., & Striegnitz, K. (2006). *Learn Prolog Now*.

IBM. (s.f.). *Rules Builder's Guide*. Obtenido de Rules Builder's Guide:
https://publib.boulder.ibm.com/tividd/td/tec/GC32-0669-01/en_US/HTML/RBGmst324.htm

Lloyd, J. (1986). *Foundations of logic programming*. Berlin.

Merritt, D. (2016). Amzi. Recuperado el 2021, de Amzi Inc:
<http://www.amzi.com/index.php>

SWI Prolog. (s.f.). *Prolog Documentation*. Obtenido de SWI Prolog:
<https://www.swi-prolog.org/pldoc/man?section=dynpreds#:~:text=Removes%20all%20classes%20of%20a,are%20reset%20to%20their%20defaults.>