



Tecnológico de Monterrey

Pokemon game

Fernando Urióstegui Peña - A01207525

Jorge Ricardo Franco Marín - A01701980

Semestre

Enero-Junio de 2021

Introduction	3
Context of the problem	3
Solution	4
Paradigm and languages	4
How it works	6
Demo battle	6
Check weakness	7
Wild encounter	7
Trainer battle	8
Results	9
Conclusions	12
Setup instructions	12
Evidence or references	14

Introduction

When you try to make a representation of the game "Pokemon" it is known that it needs a database to store N number of pokemons, moves, typings and facts such as a move being super effective, not effective, normal effective against other pokemons.

A combination between Prolog and Java, in which Prolog is a logical programming language where the user defines the relations and facts, giving an effective approach to the problem while at the same time using java to give our project a simple graphical interface for the user to have a more interactive experience.

The main purpose of the project is to use some of the most common Prolog functionalities such as backtracking and unification to help our functions gather and link the facts and statements we need in order to form lists of winning matchups or decide who would win in a small pokemon battle.

Context of the problem

Whilst we were thinking of what to do for a project on the class programming languages, my teammate and I were playing pokemon showdown. As we were playing we faced a simple issue: One of us didn't know anything about pokemon.

We realized that many people have not played past the first generation, so we aimed to recreate a pokemon which more people can feel more familiar with. For this we would need first to become experts on the topic, fortunately one of us was.

We aimed to create a demo of pokemon in which we could see what was happening, but after trying it it wasn't enough to understand pokemon, so we decided to make a function in which you can enter a pokemon name and it will tell you against what it is "Super effective", a term to describe that it is stronger than, then after this we wanted to show the stats and moves a pokemon could have, we did just that.

The actual game has some parameters for each pokemon, but due to the nature of the game and the whole RNG system behind those stats we decided to set those stats randomly. Finally we did something that you can do at the end of the games which is battle with a legendary pokemon master. This is because since we had learned about pokemon we wanted to see if we could actually beat the best trainers in a more simple way.

Solution

Paradigm and languages

As mentioned before, due to the nature of our problem and to give the user a better experience we decided to merge prolog into java, where java takes consults with queries and obtains the results we would get in prolog.

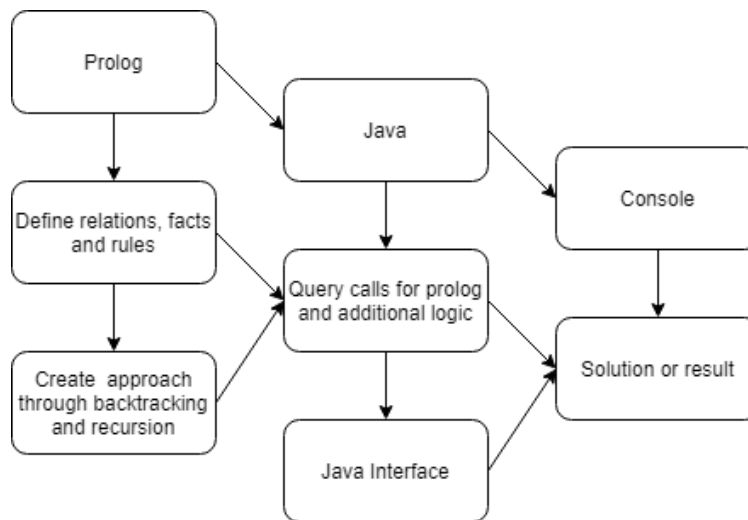


Diagram 1. Flow of the pokemon program.

The decision to use prolog instead of single programming java was due to Prologs capabilities:

- Establish facts and relations. With this we can define our knowledge base and establish facts such as a pokedex entry which includes name, id and type; superEffective which gives a relation within one type and another; moves with their typing, id and damage and so on.

```
pokedex(bulbasaur,1,grass).
```

Figure 1. Pokedex KB.

```
superEffective(fighting, normal).
```

Figure 2. SuperEffective relation.

```
grassPower(absorb,1,grass,5,7).
```

Figure 3. Moves KB.

- The use of unification. Unifying terms is extremely useful as it let's us bind variables to already known facts or values which will then be used to extract information in order to recursively find pokedex entries and move typing to check for effective moves.

```

canXWin(X,Y):-
  pokedex(X,_,A),
  pokedex(Y,_,B),
  superEffective(A,B).

```

Figure 4. Code for canXWin.

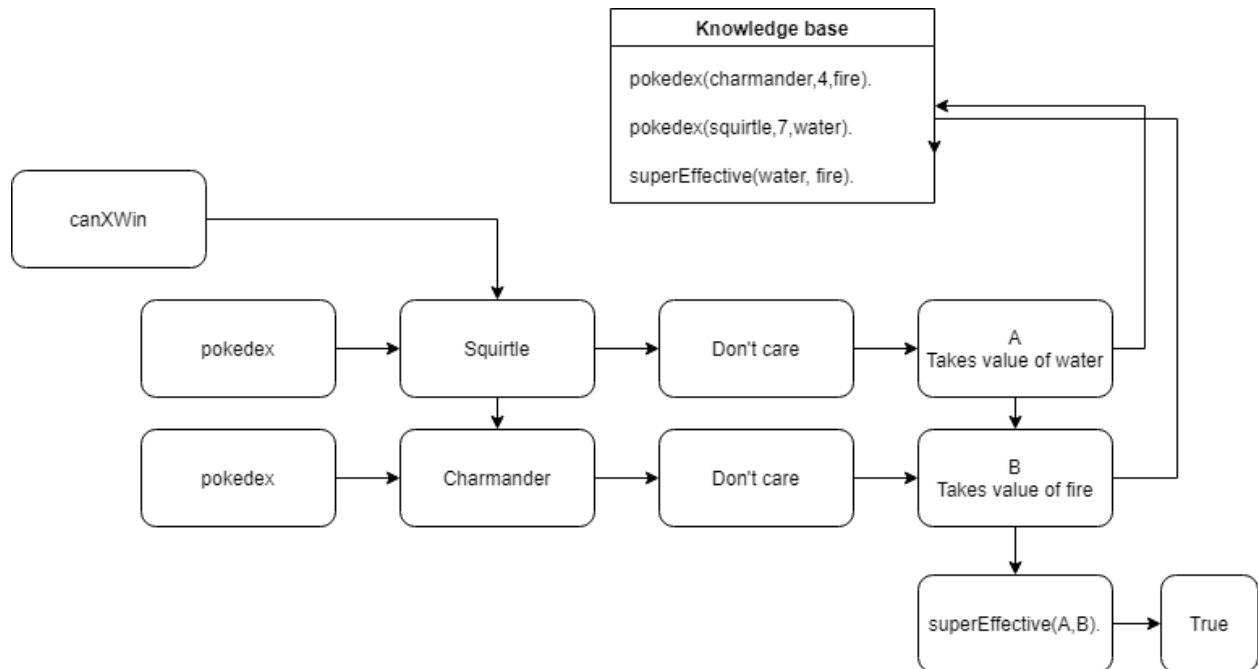


Diagram 2. canXWIN flow logic.

- Recursion and backtracking. Being able to explore the different paths and lists through these paradigms was quite helpful, not only does it allowed us to run through two lists and compare pokemons but it also gives you freedom of checking all of a pokemons strong matchups depending on the type advantage they posses over the pokedex list established previously.

Algorithm 1: doesXHaveTheSamePokemonAsY

```

1  doesXHaveTheSamePokemonAsY(X,Y):-
2    collection(X,L1),
3    collection(Y,L2),
4    checkList(L1,L1,L2,L2).
5  checkList(L1,L1,L2,L2):-
6    checkList(L1,L1,L2,L2,_).
7  checkList(_,[],_,[],true).
8  checkList(_,[],_,_,):- false.
9  checkList(_,[_|B],[E|F],[_],):-
10   checkList(B,B,[E|F],F,_).
11 checkList(L1,[A|_],L2,[C|_],):-
12   A == C,
13   checkList(L1,[],L2,[],true).
14 checkList(L1,[A|_],L2,[C|D],):-

```

```
15      A \= C,  
16      checkList(L1,L1,L2,D,false) .
```

How it works

When you open the interface we can observe 4 buttons, each one will take you to one of the functions:

Demo battle

Demo battle is one of the more complex of these functions as it requires a lot of query consults to the prolog file. Through these calls we will:

- Define effectiveness of the moves. We extract the typing of the moves being used by each pokemon and send it to a consult to prolog to check whether the move is effective, super effective, not effective or it has no effect at all on the opposing pokemon.
- Random selection of a move. Every round of attacks each pokemon will take a random move from our knowledge base based on their typing.
- Calculate the damage each move does. We extract the range of damage the random move can do and then make a few rudimentary calculations based on the base damage and effectiveness of the move towards the opposing pokemon.
- Selection of a random wild pokemon:
 - Our first pokemon “Squirtle” will take a random index from 1 to 50 and choose to battle that pokemon.
 - Our second pokemon “Charmander” will take a random index from 51 to 100 and choose to battle that pokemon.
 - Our third pokemon “Bulbasaur” will take a random index from 101 to 151 and choose to battle that pokemon.

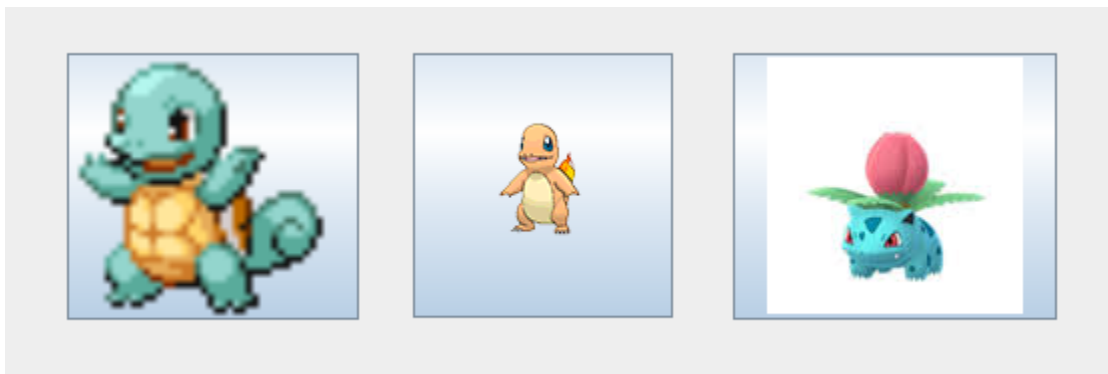


Figure 5. Demo battle graphical interface.

Check weakness

For this function we will receive the name of a pokemon desired, this will call a loop which will query the pokedex entry and take the type of the pokemon. Having this, we call another query which calls a function canXWin which determines if the pokemon has a type advantage against the 151 pokemons in the pokedex and stores them on a list which is given back to the user with all the pokemon which theoretically he can beat (this code can be found in figure 4).

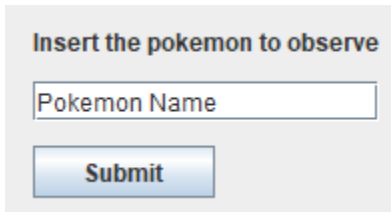
A graphical user interface for checking a Pokemon's weakness. It features a light gray background. At the top, the text "Insert the pokemon to observe" is displayed in a bold, black, sans-serif font. Below this text is a white text input field with the placeholder text "Pokemon Name" in a light gray font. Underneath the input field is a blue button with the word "Submit" in white, bold, sans-serif font.

Figure 6. Check weakness graphical interface.

Wild encounter

This function will ask the user for a pokemon that he wants to observe in the wild. Just like in pokemon games, the stats of the found pokemon are randomly calculated within a range and will take 4 random moves from our knowledge base which are based on the pokemon's type. Every single time the user asks for a pokemon it will be randomly generated but there's no chain hunting for shiny hunts.



Figure 7. Wild encounter graphical interface.

Trainer battle

- This function will ask the user for 4 pokemons, it can be the same or different, the program doesn't care if they are the same.
- After getting all the pokemons it will enter the function pokemon battle in prolog, this uses recursion to check every pokemon in the list
- It selects the first and compares it to the first of your own with the function canWin, that gives if a pokemon can beat another.
- If it yours can beat the other pokemon you kill it and it moves to the next one of the opponents list, if not it will check if the opponents pokemon can beat yours, and move accordingly
- If no pokemon can beat the other we consider it a tie and we move both lists of pokemon.
- The first to run out of pokemons loses.

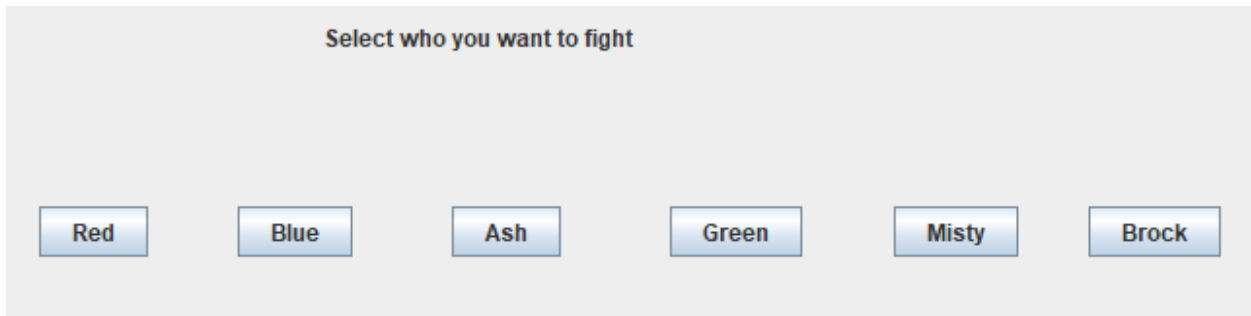


Figure 8. Trainer battle graphical interface.

```
%Kb for masters and their pokemons
collection(red, [pikachu,venasaur,charizard,blastoise]).
collection(ash,[pikachu,caterpie,charmander, pidgeot]).
collection(blue, [pidgeot,gyarados,rhydon,exeggutor]).
collection(green,[clefable,blastoise,kangaskhan,gengar]).
collection(misty, [starmie, staryu, psyduck, dugtrio]).
collection(brock,[onix,geodude,kabutops,rhydon]).
```

Figure 9. Legendary trainers and their teams.

Results

Your opponent sent out blastoise
You used surf
It's not very effective
You did 100 damage
your opponent has 90 Hp left
Your oponent used surf
It's not very effective
Enemy did 110 damage
Your pokemon has 80 Hp left
You used hydro_pump
It's not very effective
You did 140 damage
your opponent has -50 Hp left
Your oponent used hydro_pump
It's not very effective
Enemy did 140 damage
Your pokemon has -60 Hp left
Both pokemon fainted



Figure 10. Demo battle not effective.

Your opponent sent out vileplume
You used hydro_pump
It's not very effective
You did 120 damage
your opponent has 50 Hp left
Your oponent used solar_beam
It's super effective
Enemy did 150 damage
Your pokemon has 30 Hp left
You used bubble_beam
It's not very effective
You did 70 damage
your opponent has -20 Hp left
Your oponent used mega_drain
It's super effective
Enemy did 90 damage
Your pokemon has -60 Hp left
Both pokemon fainted



Figure 11. Demo battle same type.

Your opponent sent out charizard
 You used bubble_beam
 It's super effective
 You did 70 damage
 your opponent has 90 Hp left
 Your oponent used fire_blast
 It's not very effective
 Enemy did 80 damage
 Your pokemon has 90 Hp left
 You used crabhammer
 It's super effective
 You did 110 damage
 your opponent has -20 Hp left
 Your oponent used flamethrower
 It's not very effective
 Enemy did 100 damage
 Your pokemon has -10 Hp left
 Both pokemon fainted



Figure 12. Demo battle super effective.

In theory you can beat this pokemons
 [mankey, primeape, machop, machoke, machamp, hitmonlee, hitmonchan]

In theory you can beat this pokemons

[bulbasaur, ivysaur, venusaur, pidgey, pidgeotto, pidgeot, spearow, fearow, ekans, arbok, nidoran, nidorina, nidoqueen, nidoran, nidorino,

Your pokemon pikachu

Hp : 56

Attack : 88

Defense : 87

Special Attack : 149

Special defense : 72

Speed : 90

Got the powers [thunder_fang, discharge, thunder_shock, discharge]

insert the pokemon to observe

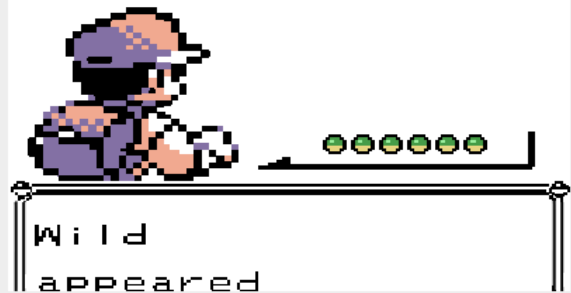


Figure 13. Wild encounter pikachu.

Your pokemon rhydon
Hp : 85
Attack : 77
Defense : 71
Special Attack : 65
Special defense : 88
Speed : 73
Got the powers [power_gem, rock_tomb, head_smash, rock_throw]



Figure 14. Wild encounter rhydon.

Choose your first pokemon to send out
pikachu
Choose your second pokemon to send out
oddish
Choose your third pokemon to send out
squirtle
Choose your forth pokemon to send out
charmander

You won the fight against blue

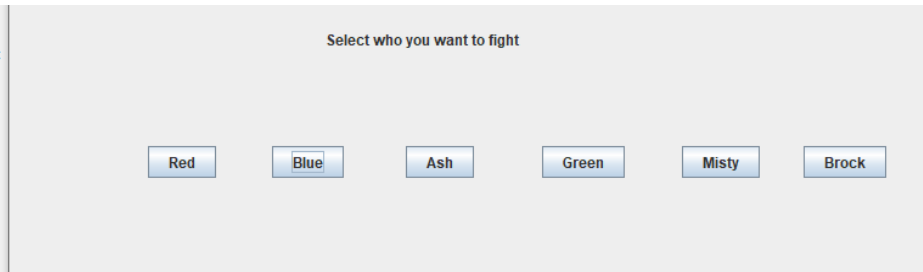


Figure 15. Trainer battle winning team.

Choose your first pokemon to send out
oddish
Choose your second pokemon to send out
oddish
Choose your third pokemon to send out
oddish
Choose your forth pokemon to send out
oddish

You lost the fight against blue

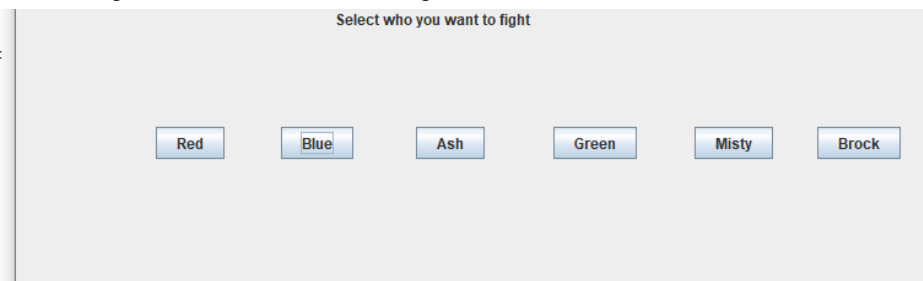


Figure 16. Trainer battle losing team.

Conclusions

When you try to do a pokemon game we learned that in fact you do need a database to store not only the pokemons, but also moves, types, strength of moves, what is effective against what, what loses to what, etc. Hence we learned that a database can be your best friend if used correctly.

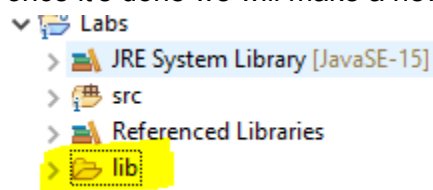
Analysing this we decided that it was a good idea to use Prolog, to define what we wanted and how we wanted, in other words do most of the logic in a “declarative way” meaning we could establish facts and relationships, we could also use the unification to make our queries, and we could use some relatively easy recursion and backtracking.

We also learned that that's as far as we knew how to take prolog, so for making things a bit smoother we also used java, java was used mostly for operations and an graphical interface, so that anyone could know what was happening as it was our premise, to help people learn about pokemon without having to play an entire game.

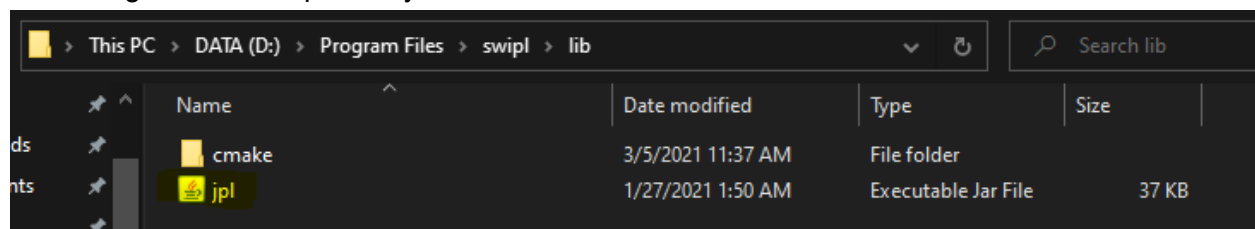
After much trial and error we discovered the strengths and weakness of what we did and it become extremely apparent how hard it would have been to make the program without a database, if not impossible, and we learned to delegate the correct things to the correct program, this way the programming languages are working for what they are best.

Setup instructions

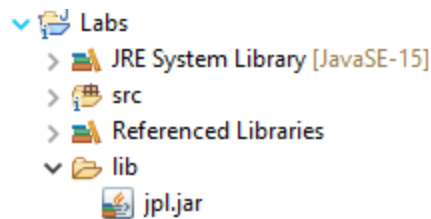
The setup for java to work with prolog was a bit complicated. For this either we need to have both prolog and eclipse(or another IDE), or download prolog libraries, after we've done that we run our IDE of preference, these instructions will be for Eclipse. We need to make a java project, once it's done we will make a new folder named lib inside the project.



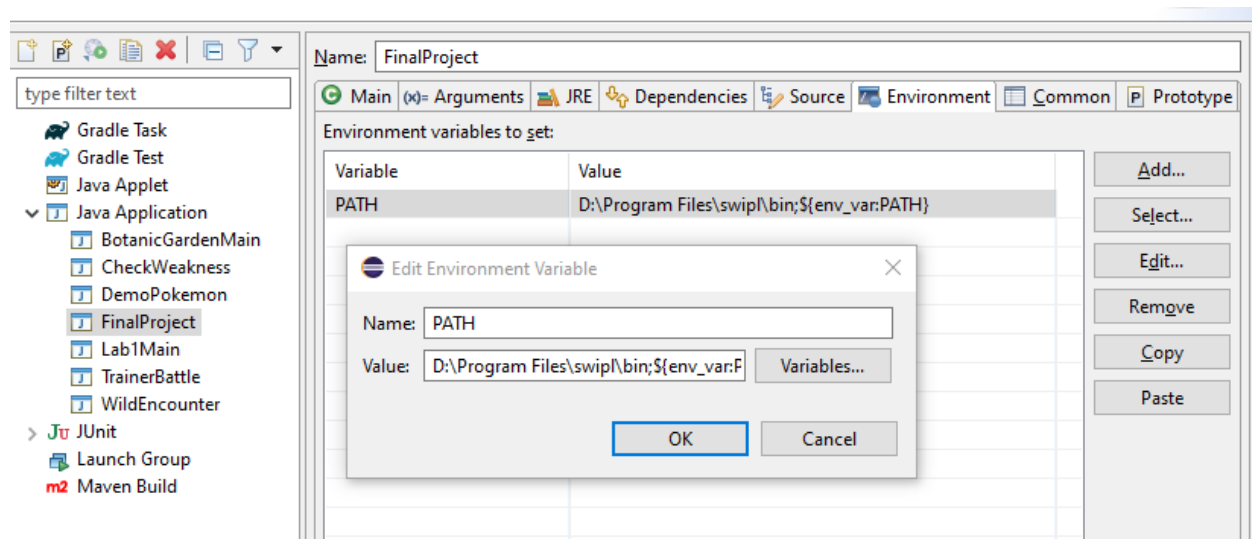
After doing that we need to add the library, if you have installed prolog you can find it under something like this, respectively.



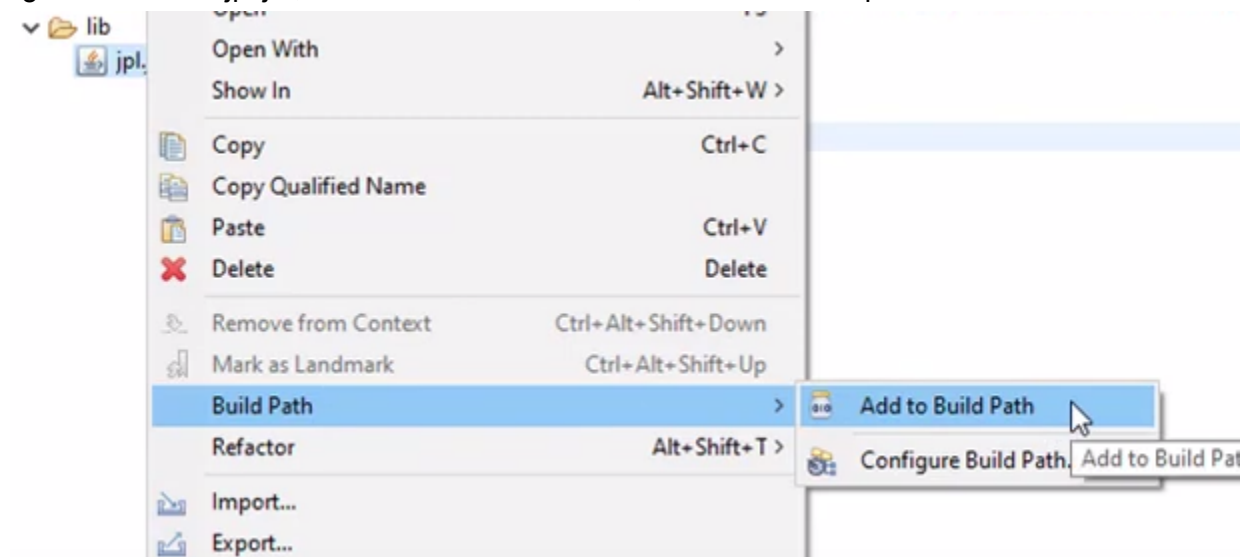
We will add a copy of that to the lib folder we had done before. It should look like this.



Then we will have to add how we want to run the program, and to add the prolog library we will need to create an environment variable, to do this we will need to go to run configurations and to environment how we will add the name PATH, and the value will be something like that. D:\Program Files\swipl\bin;\${env_var:PATH}, the changes anyone will have to change would be the path before the “;”, that path will be where you have your bin folder respective to prolog, almost the same as for the jar we did before, should look something like this.



Finally we will need to add the library to the build path we just created, to do that we will press right click on the jpl.jar, we added to the lib folder, and select build path



After doing all this the program is ready to run, click and run class GUI to see the interface and start using the program.

Evidence or references

<https://www.youtube.com/watch?v=yaitNTFvF7M&t=913s>