CMPE260 - Farm Simulation Project Report

Üstün Yılmaz, 2023400108

Apr 3, 2025

Abstract

In this project report; the background, the implementation, and the conclusion of the assignment "Logic-Based Farm Simulation Project" will be analyzed and evaluated.

1 Introduction

The course project of CMPE260 - Principles of Programming Languages, Spring 2025 involved implementing a logic based farm simulator in the logic programming language, Prolog. The process of implementation was done in three main steps:

- Finding the necessary tools to implement the project
- Thinking about the way to implement the project
- Implementing the project in code form

2 Finding the Necessary Tools to Implement the Project and Implementation - Ideation Process and Workflows

The first step in this ideation process was to choose an interpreter to suit my needs in coding the project. As we have discussed with the professor in the lecture, SWI-Prolog was the most suitable to implement a Prolog project of this level, so I decided to set it up.

After the interpreter was good to go, it was time to think about how to handle each type of predicate/query given in the project outline. I researched SWI-Prolog's website to find out about all necessary builtin predicates like findall, aggregate_all, get_dict etc. and analyzed their purposes thoroughly. I also checked out cmpefarm.pro to see how the secondary "builtin" predicates were implemented, as I was also going to need these in the project.

Then I planned how I would implement each type of predicate we were asked to implement, and came up with these solutions:

1. **agents_distance:** Here, I would get the coordinates of both agents using get_dict and substitute in the given formula for Manhattan distance.
2. **number_of_agents:** Here, I would use aggregate_all to count how many agents there are in the farm.

3. **value_of_farm:** Similar to above, I would use aggregate_all but weight each finding with the value of that agent.

4. **find_food_coordinates:** Here, I utilized findall to return the locations of the food with two cases, the agent being carnivore and the agent being herbivore.

5. **find_nearest_agent:** Here, I utilized findall to aggregate all agents that are not the current agent in a list/dictionary, and used keysort to extract the one with the minimum.

6. **find_nearest_food:** Here, I combined the logic of the previous two functions, aggregating all objects for herbivores and aggregating all non-wolf agents for carnivores. Actually, the can_eat predicate did most of the job for me.

7. **move_to_coordinate:** Here, as no approach was specified in the project outline; I decided to write an iteration based BFS that is limited in iterations by DepthLimit (I also tried depth-first search but it was **EXTREMELY INEFFICIENT**. And then I forced it to extract the solution with the smallest number of moves so that my code is more efficient (Weirdly, the testcase in the project outline is nowhere near optimal.).

8. **move_to_nearest_food:** This predicate was no more than just combining the last two predicates, as its name suggests.

9. **consume_all:** This one was the hardest of them all, since the added logic of making a children and generating/consuming energy. I used the same logic in find_nearest_food to get to the food, and utilized make_series_of_actions that was given in cmpefarm.pro. And I implemented two separate helpers, one to handle the inner loop logic of this predicate and another one to handle the cases where eating the food is not possible.

3 Conclusion - Challenges Encountered and Lessons Learned

The first and maybe the biggest challenge that I encountered during the project was to use which approach to perform an efficient movement to a specific coordinate in the predicate move_to_coordinate. I first implemented an iterative DFS (Depth-First Search) that was limited by the DepthLimit, but the I saw it performed really sluggish at depth limits that were greater than 10. So I researched the internet for better approaches in Prolog and found a BFS (Breadth-First Search) implementation on StackOverflow[1] and inspired by that code, worked on writing my own BFS implementation tailored specifically for this project.

Another challenge that I encountered was that my code sometimes differed from the outputs given in the project outline. But as I kept analyzing farm.txt and the way the predicates were defined in cmpefarm.pro, I realized that predicates like move_to_coordinate were not deterministic and could have different results depending on the implementation. That is where I decided to tiebreak my BFS by shortest distance.

The main lesson that I learned from this project was that Prolog can be extremely useful when it comes to implementing logic based programs that utilize recursive structures. That said, it was a language that I was not really familiar with as we had to deal with many builtin methods we did not encounter in our lectures before, but all in all, it was a rewarding experience to add another programming language, and a whole new programming paradigm called logic programming to my arsenal.

References

[1] StackOverflow, *Breadth First Search in Prolog*, 2015. Available at: https://stackoverflow.com/questions/34082799/breadth-first-search-in-prolog/34094494#34094494