EECS-395 Assignment 4

Out: Wednesday, May 20

Due: Wednesday, May 27, 11:59PM

# Overview

In this assignment, you will implement need satisfaction in the problem solver. Here’s the basic idea:

* You will implement a task, satisfy\_needs, in the problem solver that loops, repeatedly satisfy\_a\_need
* You’ll write a set of strategies for satisfy\_a\_need that satisfy particular needs using particular actions and objects
* That will mean the problem solver will be confused which strategy to use, and will, have to give up and choose randomly
* To make it better, you’ll give it a meta-strategy to choose strategies by scoring the needs they satisfy.

For this assignment, you don’t have to worry about death or addition.

# Getting started

Grab the latest version of the game code from Canvas, if you haven’t already. Then go into emacs and open the file Assets/Problem solver/need\_satisfaction.prolog. Note that this is a different file from the one used for the previous assignment – it’s in the Problem solver directory, rather than the Concerns directory. The one in concerns is still there, but the system will no longer start up that concern code. The code of satisfaction\_level and increase\_satisfaction is still there, however.

# Making a loop

The file includes code to start up a task, initialize the timers for tracking need levels, and run the magic task satisfy\_needs, which takes no arguments. Start by writing a strategy for satisfy\_needs that repeatedly calls satisfy\_a\_need (again, with no arguments), over and over again, forever. You will need to use tail recursion for this.

# Satisfying a need

Now write a number of methods for satisfy\_a\_need that each:

* Go to an object (e.g. the refridgerator)
* Print some pithy phrase (e.g. “munch munch munch”)
* Say call(increase\_satisfaction(*need*, *amount*)), filling in the appropriate values for *need* and *amount*.

When you run this, you should see the characters randomly running strategies, since you haven’t told it any way of intelligently choosing between them.

# Meta-level reasoning

Now you want to add a meta-strategy for choosing between the strategies generated in the previous part. That means you’re going to need to write a rule that looks something like:

strategy(resolve\_conflict(satisfy\_a\_need, StrategyList),  
 ChosenStrategy) :-  
 *do something here to choose the strategy*.

The code after the :- has to look at the strategies in StrategyList, and choose one.

This will be easier to do if you refactor the code from the previous section somewhat. Before, we had you write:

strategy(satisfy\_a\_need,  
 begin(…*go to the refrigerator and eat* …)).

Which would mean the StrategyList above is a list of begin expressions, which the meta-strategy would have to somehow look inside of to figure out which are satisfying hunger and which are satisfying some other need. If instead of writing it this way, you can write:

strategy(satisfy\_a\_need,  
 satisfy(hunger, $refrigerator)).

I.e. “you can satisfying a need by satisfying hunger using the refrigerator.” You can then specify how to satisfy hunger using the refrigerator with another strategy:

strategy(satisfy(hunger, $refrigerator),  
 begin(…*go to the refrigerator and eat* …)).

i.e. “to satisfy hunger using the refrigerator, go to the refrigerator and eat.”

The advantage of writing it this was is that the StrategyList above won’t be a list of random begin expressions, it will be a list of tasks, all of the form satisfy(*Need*, *Object*), and so you can:

* Take the *Need* and *Object*
* Compute a score for from it
* Use the arg\_max predicate to choose the satisfy expression with the largest score. Arg\_max is defined as:  
    
   arg\_max(*Answer*, *Score*, *CodeToGenerateScoresAndAnswers*)  
    
  for example:  
    
   arg\_max(satisfy(Need,Object),  
   Score,  
   ( member(satisfy(Need,Object),   
   StrategyList),  
   Score is *whatever* )).

You can use any of the scoring systems discussed in the slides on need-based AI or in the Zubek article.

# Turning it in

Just upload the need\_satisfaction.prolog file from the Problem solver directory (not the Concerns one). That’s all you need to turn in.