COS 470/570 - Program 2

A Basic Agent

Assigned: 2/11 Due: 2/25

It has been my experience that this program is harder for students than it seems. **START EARLY** so you can get help as you need it!

1 Introduction

For this assignment, you will create a basic agent and a simple agent simulator similar to the one described in your textbook.

The simulator will allow one or more agents (only one, for this assignment—but think ahead!) to move about in a simulated world. It will be in control of the agents, in the sense that it can stop and start them, etc., and only it can move an agent. However, it will not be in control of what the agent wants to do—that is, each agent will have its own *agent program* that interprets the percepts given to it by the simulator, decides on an action, and informs the simulator of the desired action. Consequently, this assignment requires:

- a simulated world in which one or more agents can move about;
- an agent program to control each agent;
- a means to represent each agent's sensor input as percepts for use by the agent program;
- a means to represent what action the agent wants to take for use by the simulator when determining what the effects of the action will be; and
- a performance measure—or more than one—to evaluate the agent's behavior and solution to problems it is assigned.

You will use your simulator to test the performance of two types of agents (described below), and you may use the simulator for future assignments. Therefore, you need to think carefully about your design before you start coding. In particular, your design should be able easily to allow different agents to be tested, different worlds to be simulated, and different performance measures to be used.

There are three things to keep separate in this assignment: the agent, the agent program, and the world. It is important that you cleanly separate the agent and agent program from the world. The agent program can only know what the agent's sensors tell it about the world. It cannot know, for example, its *actual* location or the actual location of obstacles, etc. Only the world knows that. Similarly, what the agent program chooses to do in the world only affects the simulated world to the extent the world lets it: the agent may try to move, for example, but the world may not let it (e.g., if there is an obstacle in the way).

I would strongly suggest that you write your code in an object-oriented fashion using CLOS, with one class for the simulator, one for the world, one for the agent, and one for the agent program. This cleanly separates the concerns of the program:

- The simulator will be concerned with the overall control of the simulation, with data collection, and possibly with statistical analysis.

- The world will be concerned with where the objects (including the agent(s)) are, where the boundaries of the world are, etc., as well as with permitting some actions (e.g., movement) and prohibiting others (e.g., movement into an obstacle).
- The agent will be concerned with a model of the sensors of the agent and with asking the world to move the agent and take other actions. For example, at some time you may want to model uncertainty in the agent's sensors, as would be the case in a real robot; this would be done in the agent's sensor functions.
- The agent program is concerned with sensing, thinking, and taking action. It will, for some types of agent, keep track of what it thinks is the agent's state.

This object-oriented design has the great benefit of allowing you to have several kinds of agents and agent programs simply by creating new classes based on a generic agent class sand a generic agent program class. For example, in this assignment, the agent's sensors will return accurate information about the world. However, at some future time, you may want an agent that models noisy sensors; this can be done by simply creating a new class based on the agent class, then defining a different sense function for that class.

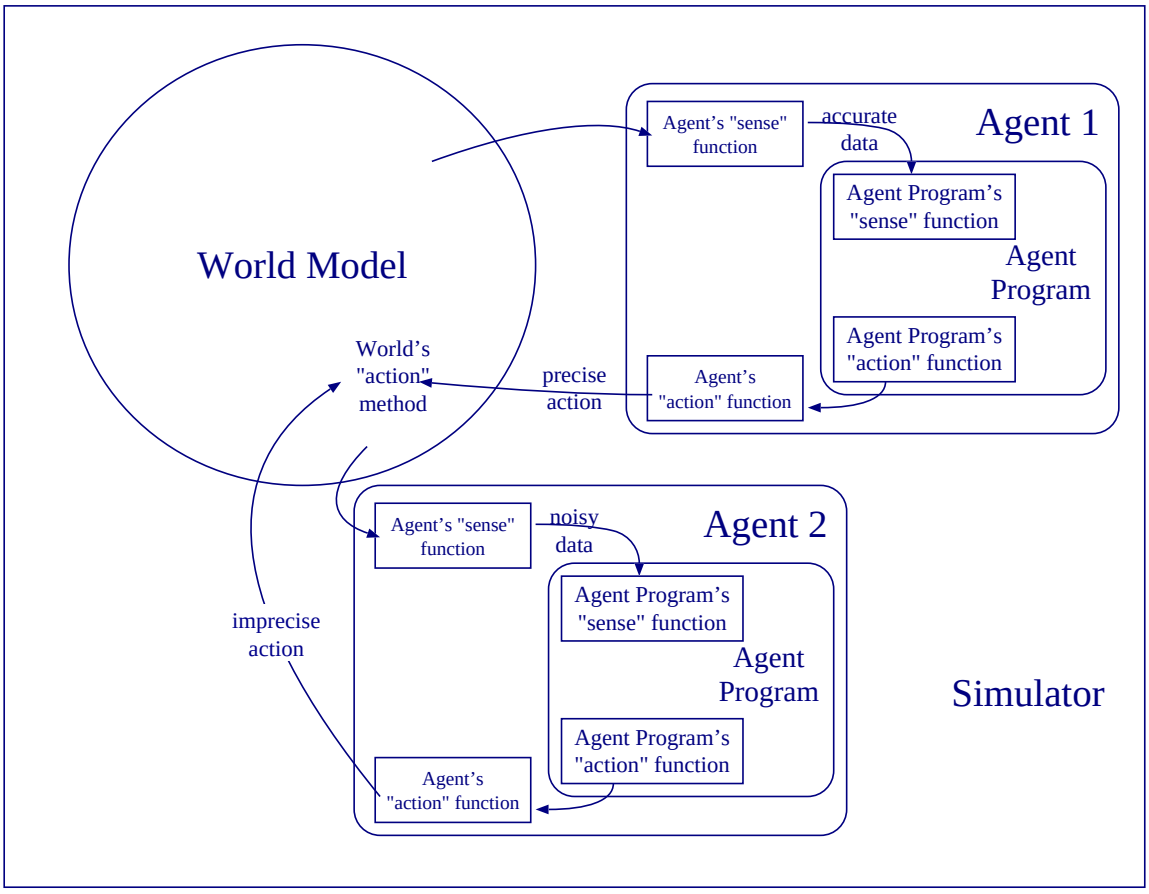


Figure 1: A suggested simulator design.

Figure 1 shows this suggested design, with two different agents.

The program may seem simple, but remember that you are also learning a new language from a different paradigm. Be sure to start early and give yourself plenty of time to both design the program and implement it in Common Lisp. **Design it first**—then code it. You should be able

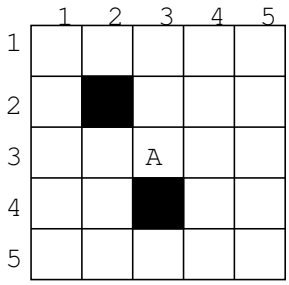
to design the program without regard to the language you'll use to code it. Coding is simply the translation of your design into a concrete form the computer can run.

Remember as you are implementing the simulator that Common Lisp facilitates writing programs in a top-down manner. You can do this using dummy functions ("stubs") which return some constant value or some value input by the user (using the *read* or related Lisp functions). You can then test the high-level functions. After the general mechanism works, you can replace the dummy functions with Lisp code to automatically generate the proper result, and so work your way down to the lower-level code needed.

2 The Simulator

2.1 The Environment

For this assignment, the agents will move around in a world that can be represented by a five-by-five grid:



The world is surrounded by a wall and there may be obstacles (shown as black cells, above) in the world. These obstacles take up an entire space (cell) on the grid. Although you will want to be able to change the world between trials of your agent program, you should not worry about the world changing during a particular trial.

You can represent the world any way that makes sense to you. One possible representation is to keep a list of locations of obstacles and assume that all other locations are clear. It will probably be easier for you at this stage (and you will get more practice with list manipulation, which will help you in the future) if you store these values in a list or lists.

You will also need to represent the agent in the world. All that you really need to know about the agent is its location and the direction that it is facing. For example, you could simply represent the agent with a triplet that gives x and y coordinates and heading. Keep in mind, however, that at some point you might want more than one agent active in the world at once.

2.2 The Agents

In this assignment, you will simulate two different types of agents:

1. a simple reflex agent; and
2. an agent that keeps track of the world through a history or other mechanism.

Each agent will have following sensing capabilities:

- sense an obstacle in the grid space directly in front of it;
- and sense that it has bumped into an obstacle, and tell whether the obstacle (the bump) was on the left, right, front or back.

These two sensing capabilities are concurrent; the results of both should be provided to the agent program in its percept string (or list, or whatever). Note that the bump sensor senses the result of the previous action on the part of the agent.

Both agents will be able to perform the following actions:

- turn left
- turn right
- move forward
- move left
- move right

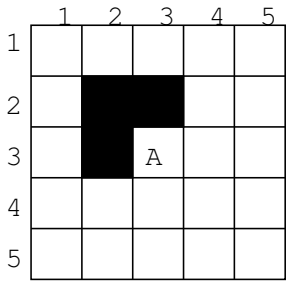
The agent may also take no action during a "cycle"—i.e., you may add a no-op (NOP) action to those listed above.

If the agent attempts to move into a space where there is an obstacle, it will feel the obstacle but will not be able to move from its current position.

2.3 The Tasks

You should try to get your agents to perform the following tasks:

- Go to a corner of the grid in a world without obstacles
- Go to a corner of the grid in a world with obstacles, but in which no two obstacles can be adjacent
- Go to a corner of the grid in a world in which obstacles may be adjacent. For this there may be no more than three adjacent obstacles. However, the obstacles can have the configuration shown below. Only the agent that keeps track of the world needs to perform this task.



Note that **the agent must remain in a corner** in order for it to count as a success—the simulator can't just stop the agent when it notes that it has entered a corner. The agent has neither a stop action nor a corner sensor, so you'll need to figure out some way to make it stay in a corner once it finds one. Your simulator may stop the simulation after it detects that the agent has remained stationary for some number of time units.

2.4 Performance Measure

You will need to implement some performance measure for the each task. The simplest performance measure is whether or not the agent found a corner. You can simply allow the user to evaluate the performance using this measure. This is an adequate performance measure for this assignment. You should also feel free to implement a better performance measure; there's always room for a better grade, after all!

3 See How They Run

Okay, bad pun. But you will need some way of showing how your agents performed. This can be as simple as a list of the actions the agent took along with the (x, y) coordinates the agent visited (and its heading), or it can be as fancy as a nice diagram of the 5×5 grid. Your choice, but it should be *readable* by me!

Note that Lisp has several ways of doing this, depending on which version you are using and how fancy you want to get. The easiest way might be an ASCII diagram of the world, with characters representing open space, obstacles, etc. Or you can have Lisp call (or even produce) a Java applet, or even OpenGL calls to do a 3D interface. (A bit overkill for this assignment!) If you are using the Windows version of Allegro CL, there is likely built-in drawing functions in the Lisp image.

4 Discussion

For all of your programming assignments in this class, you will have a write-up in which you will discuss what you have done. Each of these will begin with an overview of the design of your program. For this assignment, be sure to include:

- details of the overall simulator and agent architecture
- information about the interface(s) between the components of your system
- how you ensure that the world knowledge and agent knowledge will be separate
- how you carry out certain important functions (e.g., updating the world, the performance measure)
- the programs used to get the agents to perform their tasks

There should also be enough comments/documentation in your code that I can understand how your program works. In addition, you should include a function called "run-simulator" that takes no arguments that I can call to run your simulation after it is loaded. This way, I can test your simulation myself to get a sense of how it runs.

For this assignment, you should also answer the following questions:

1. What did you have to change as you changed agent types and tasks?
    (a) Do these changes tell you anything important about the nature of these types of agents?
    (b) Did you find flaws in the design of your simulator that should be changed before you attempt to use it for other agents and tasks?
2. Do you think your agents would "scale-up"?
    (a) Would your agents be able to perform the tasks on a larger grid?
    (b) Would your agents be able to perform the tasks with more obstacles?

5 To turn in:

1. Your well-documented code
2. Example runs for each agent type performing each task
3. Example runs showing off any features of your program that you would like me to see
4. Your write-up

You will turn these in electronically via Synapse. For the runs and code, plain text is fine, although if you have a fancy way you want to format them, then turn them in as a PDF or .doc file. For your write-up, either PDF or .doc is fine. (LaTeX users—and I hope most of you will be at some point—use `pdflatex` to generate PDF from .tex files or `ps2pdf` or Adobe Acrobat to generate PDF from PostScript files.) If you cannot handle these requirements, see me before the due date.

6 Grading

50% of your grade will be based on the programming portion of this assignment. You should be sure to point out important features of the program in your write-up. I will look for the following:

- working code for the simulator and the required tasks for both agents
- well-designed agent programs
- a well-designed architecture
- well-designed code (at least for a beginning Lisp user)
- well-documented code

50% of your grade will be based on the write-up of the assignment. I will look for the following:

- thoughtful answers to the questions above based, in part, on your experiences with the program for this assignment
- a well-organized, well-written paper that is free from typographical, spelling and grammatical errors