University of Maine
School of Computing and Information Science

COS 470/570: Introduction to Artificial Intelligence

Agents & Search Programming Assignment

Spring 2019

Assignment: 1/31/2019 **Due: 2/14/2019**

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!

In this assignment, you will create a simulator for simple robot agents, then create several types of simulated agents and test them in the simulator. You will also apply state space search to a second domain.

Robot World

The simulator

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 which action the agent wants to take that the simulator can use 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 the agents described below. Since you *may* use the simulator for future assignments, 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.

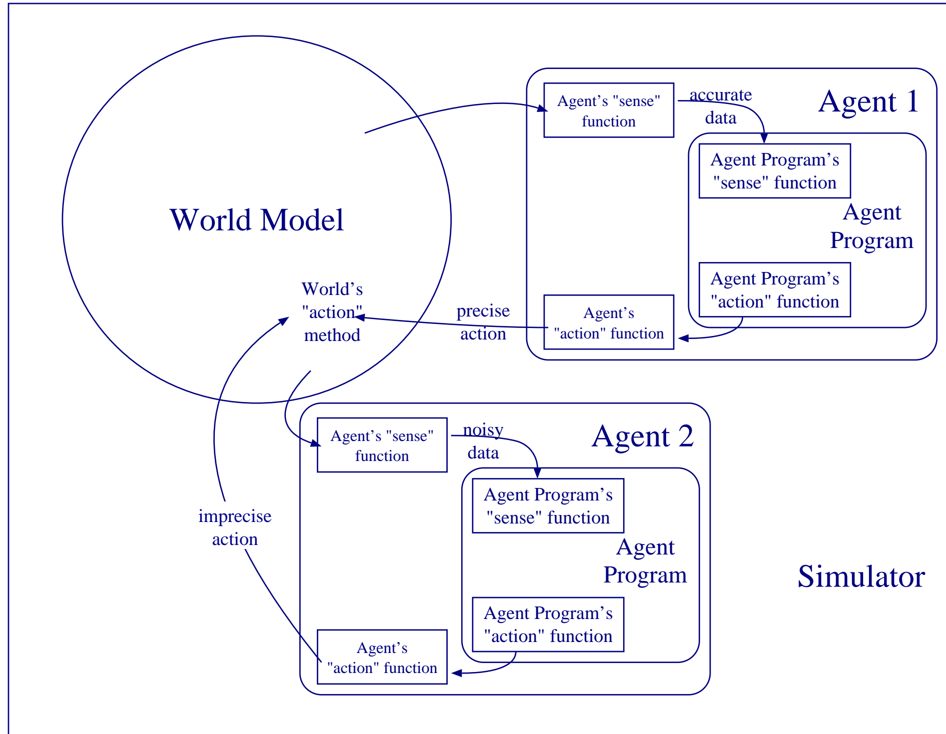
There are three things to keep separate in this assignment: the agent, the agent program, and the world. It is important that you cleanly separate the agent and agent program from the world. The agent program can only know what the agent's sensors tell it about the world. It cannot know, for example, 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.

This diagram shows this suggested design, with two different agents:



The program may seem simple, but remember that you are also learning (or continuing to learn) 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 `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.

The simulated world will be a grid, e.g.:

	1	2	3	4	5
1					
2					
3			A		
4					
5					

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 where the agent is 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 may want more than one agent active in the world at once.

The agents

The agents themselves will be simple robots that accept percepts, make decisions, and take action. Each is controlled by an agent program, and this is what will change from robot to robot. Each cycle, the simulator will give the agent a percept, receive the agent's desired action, and then simulate the result of the action, e.g., by moving the agent. If the agent is implemented as an object, this can be done by calling a method of the agent instance, for example.

An agent has a forward-looking sensor that can see if the space directly in front of it is clear or not, and it has bump sensors that can detect if it just bumped into something as a result of its last action. Thus, the percept the simulator gives the agent needs to contain these two pieces of data at once. E.g., if the percepts are represented as lists of the form:

```
(front-sensor front-bump left-bump right-bump rear-bump)
```

then the percept:

```
(t nil nil t nil)
```

would mean that there is nothing directly in front of the robot and that it bumped into something on the right as a result of the preceding action. You are, of course, free to represent the percepts however you choose.

The agent has a fairly limited set of actions it can take: it can move 1 cell forward/left/right/backward or it can turn 90° left or right. The agent will pass an action request to the simulator each cycle. It can also do nothing, which means it sends a "no-operation" (NOP) action request to the simulator. Action requests can be represented any way you like. A simple example might consist of sending a single symbol, e.g., from the set `{F,L,R.B,N,TR,TL}`, or a more structured form might be something like:

```
(MOVE x), (TURN x), NOP
```

where x is a symbol representing a direction.

Internally, the agent needs to receive a percept somehow, pass that along to the agent program, then take the action the agent program suggests and pass it back to the simulator. (And, yes, you *could* just use the agent program directly without having a separate agent; however, there are often reasons to separate the two, so that's what we'll do here.)

Agent programs

You will create and test several different agent programs:

1. a simple reflex agent;
2. a reflex agent that can keep track of the world and it's own actions (i.e, a model-based agent);
3. a hill-climbing agent;
4. a search agent that uses breadth-first search; and
5. a search agent that uses A*.

For 1 and 2, the goal will be for the agent to find a corner and remain there, no matter where the agent is placed in the world. Note that neither of these know anything about the world other than what their sensors (percepts) tell them. In particular, they have no way of knowing where in the world they or their goal is at any time, nor can they recognize even that they *have* a goal: they just behave according to their reflexes. If the agent attempts to move into a space where there is an obstacle, it will feel the obstacle via a bump sensor, but will not be able to move from its current position.

For 3, the agent *will* need to be told where its goal is, and it will need to keep track of where it is. The choice of the agent's heuristic is up to you. The agent, however, will *not* know where obstacles are ahead of time—that is, it won't have a map of the world.

For 4 and 5, the agent will also need to know were its goal is and where it is to begin with, *and* it will need to be given a map of the world showing the obstacles and boundaries. Note that search will happen "in the agent's head" for these agents, as per our discussion of a general search agent in class.

Performance measure

You will need to implement some performance measure(s) for each task and agent. The simplest performance measure for a reflex agent, for example, is whether or not the agent found a corner. A better measure would take into account how the agent's path compared to the optimal path. (Note that you can easily find the optimal path length using your A* program.) Similarly, for the hill-climbing agent, you could measure the path taken by the agent compared to the optimal path.

For the two search agents, you need to somehow measure the search work they did, since they should both traverse optimal paths in this domain. A good approach here is to note the number of search nodes they expanded or created, or how many edges of the search space the algorithm traversed. You should also consider keeping track of an estimate of the maximum space used; this can be done by tracking the maximum size of the searches queue or open list, for example.

Simulation runs

Reflex agents

Run each agent on three different kinds of 25 $\times$ 25 grids, with the goal being to end up in a corner and stay there:

1. worlds without obstacles;
2. worlds with obstacles, but in which there are no “fake corners”, neither formed by obstacles nor by an obstacle and a wall; and
3. worlds where fake obstacles occur.

Run the agents multiple times for each, varying where the agent begins, the number and location of obstacles (for the last two kinds of worlds), etc.

Note: **The agent must remain in a corner** in order for a run 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.

Hill-climbing agent, search agents

Run the agents on the same kinds of worlds as above, varying where the agent and the goal are located each time.

See how they run

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! (For the toy domain, you’ll need to figure out some way of displaying what is happening.)

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 some Lisps 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.

Search in other domains

Choose a domain in which to implement searches (BFS and A*) to compare with each other and to your robot world. This should be one of the traditional AI toy search problems:

- missionaries and cannibals
- towers of Hanoi
- cryptarithmetic
- traveling salesman
- 8-puzzle

- 8 queens
- water jug problem
- monkeys and bananas
- blocks world problems

You should pick a problem that will help you to understand more about breadth-first search and A*. You will be asked to justify your selection in the write-up for this program. You will also need to vary the problem for test runs, so you should choose a domain where the initial and goal states can be easily varied to create different problems. Choose a reasonable heuristic for the A* search.

Discussion

For this assignment, you will turn in not only the code, but a write-up that describes and discusses your results. This will need to include

- details of the overall simulator and agent architecture, including information about the interface(s) between the components of your system, how you ensured that the world knowledge and agent knowledge were separate, etc.;
- thorough discussion of your agent programs;
- discussion of your simulation runs and their results;
- discussion of your second domain, including how you had to change your programs to work in the domain and the results of running your programs;
- comparison of the different agent programs' performance in the robot world; and
- comparison of searching and search results in the second domain and in the robot world.

Your write-up should address at least the following questions:

1. What did you have to change as you changed agent types and tasks?
   (a) Do these changes tell you anything interesting about the nature of these types of agents?
   (b) Did you find flaws in the design of your simulator that would have to be changed before you attempt to use it for other agents and tasks?
2. Would 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 (lots) more obstacles?
3. What is your second domain? What operators did you use, what is different in the test runs, what heuristic functions did you use, and why did you choose it?
4. How did the results compare?

- Compare the number of nodes created for breadth-first search and for A*. You will want to look at both the average number of nodes created for all 20 runs and the number created for each individual run. You will need to do statistical analysis of the results; I'd suggest something like a paired t-test, which Excel should be able to do for you.[1] You should report your results along with a the confidence you have in them, expressed as a p-value. For example, "BFS produced more nodes than A* ($p < 0.05$)," which means, roughly, that you're 95% certain of the conclusion.[2] Do this for run times, too. How can you explain the results?
- Compare the number of nodes created and the run times for A* when different heuristics are used. You will want to look at both the average number of nodes created for all 20 runs, and you may need to look at the number created for each individual run. Compare run times as well. Again, use statistical analysis to bound the confidence in your conclusion. How can you explain the results?
- Are there other variables that you would like to compare? If so, what are they and how would you run tests to compare them?

5. How did your search agents compare to your reflex agents?
6. What have you learned, or demonstrated, that might be useful when implementing AI systems in the real world?

A good organization for your write-up is something like:

- Introduction/Background – what are you trying to do? Etc.
- Methods – include here any discussion of your statistical techniques, programs, etc., that will be useful. (You must tell me the statistical basis for your conclusions.)
- Results – describe the data obtained.
- Discussion – conclusions, etc.

That is, the general format for a scientific paper.

There should also be sufficient comments/documentation in your code that I can understand how your program works. In addition, you should include a function, `run-simulator`, 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.

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

[1] If it can't, and you don't have access to any other statistical software, then see me—we have a copy of a Lisp-based statistics package locally.

[2] Really that there is less than 1 chance in 20 that the results you obtained are due to chance.

4. Your write-up

You will turn these in via Blackboard; I'll put a "portal" up for your assignment before the due date. 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 file. Your write-up **must** be in PDF form. LaTeX users can use `pdflatex` to generate PDF directly.

Grammar and spelling, as well as tone (i.e., this is a more or less formal paper, so no colloquialisms, etc.), will count toward your grade.

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