Lab 8

Due: Tuesday 04/16/2024 @ 11:59pm EST

The purpose of labs is to practice the concepts that we learn in class. To that end you will be writing java code that uses a game engine called Sepia to develop agents that solve specific problems. In this lab we will be maze solving in a stochastic world using the Value Iteration algorithm.

1. Copy Files

Please, copy the files from the downloaded lab directory to your cs440 directory. You can just drag and drop them in your file explorer.

- Copy `Downloads/lab8/lib/lab14.jar` to `cs440/lib/rl-viter-maze.jar`.
  This file is the custom jarfile that I created for you.
- Copy `Downloads/lab8/data/labs/rl` to `cs440/data/labs/rl`.
  This directory contains a game configuration and map files.
- Copy `Downloads/lab8/src/labs` to `cs444/src/labs`.
  This directory contains our source code `.java` files.
- Copy `Downloads/lab8/viter.srcs` to `cs440/viter.srcs`.
  This file contains the paths to the `.java` files we are working with in this lab. Just like last lab, files like these are used to speed up the compilation process by preventing you from listing all source files you want to compile manually.
- Copy `Downloads/lab8/doc/labs` to `cs440/doc/labs`. This is the documentation generated from `rl-viter-maze.jar` and will be extremely useful in this assignment. After copying, if you double-click on `cs440/doc/labs/rl/maze/index.html`, the documentation should open in your browser.

2. Test run

If your setup is correct, you should be able to compile and execute the given template code. You should see the Sepia window appear.

```
# Mac, Linux. Run from the cs440 directory.
javac -cp "./lib/*:." @viter.srcs
java -cp "./lib/*:." edu.cwru.sepia.Main2 data/labs/rl/maze/Maze.xml

# Windows. Run from the cs440 directory.
javac -cp "./lib/*;." @viter.srcs
java -cp "./lib/*;." edu.cwru.sepia.Main2 data/labs/rl/maze/Maze.xml
```

Task 1: Value Iteration (50 points)

In this task, I want you to implement the value iteration algorithm to calculate the utility values for every state in our tiny world (this is the world example from class). The trouble is that now whenever we select an action (i.e. moving in a cardinal direction), we may not actually end up going in the direction we intended. Please complete the following method:

- `src.lab4.agents.ValueIterationAgent.valueIteration`: This method is where you should implement the value iteration algorithm. Remember, the value iteration algorithm is an **offline** algorithm where the agent has perfect knowledge of the transition model, the rewards, and the world. Your algorithm should calculate the true utility value for every state (i.e. coordinate) in the world. Please refer to the value iteration algorithm we discussed in class, and keep in mind that when running the bellman equation: we only have to consider transitioning to future states when the current state is nonterminal! This is just a long way of saying that the utility value of every terminal state is the reward you get in that terminal state.

Notes

- Testing this agent is really easy: I have implemented (and imported the `TransitionModel` and `RewardFunction` for you as static classes with static methods. Using the rewards and transition probabilities in the implementation (which match the ones from the lecture), the lecture slides have the true utilities that you should be getting! So, whenever you want to check, print out your utilities and compare them against what they should be in the slides.
- You may create whatever helper methods you want in order to accomplish this goal, however I am not sure that you will need to create any in `LogicAgent.java`.

Task 2: Submitting your lab

Please submit `ValueIteration.java` on gradescope (just drag and drop in the file).