CS 7750: Solutions to bonus assignment 1

Chanmann Lim

September 16, 2014

Exercises

2.9 Implement a simple reflex agent for the vacuum environment in Exercise 2.8. Run the environment with this agent for all possible initial dirt configurations and agent locations. Record the performance score for each configuration and the overall average score.

Result:

```
 [ Dirty* | Dirty ] = 1998
 [ Dirty* | Clean ] = 2000
 [ Clean* | Dirty ] = 1999
 [ Clean* | Clean ] = 2000
 [ Dirty | Dirty* ] = 1998
 [ Dirty | Clean* ] = 1999
 [ Clean | Dirty* ] = 2000
 [ Clean | Clean* ] = 2000
--------------------------
 Average score = 1999.25
```

Implementation description: The overall average score of the agent is the mean of the performance scores of the agent given 8 different possible initial dirt configurations and agent locations and to record the performance score for each dirt configuration of the agent, a performance-measuring environment simulator must be setup.

Performance measuring simulator - "PerformanceMeasure" class is able to evaluate an "Environment" given an agent function and measure the performance score for each time step of the agent over the agent's lifetime (in this case is 1000 time-step).

"Environment" class contains current state/percept of the vaccum-cleaner world and its state can be changed by action of the agent in each time step. For instance, when the agent perform "Suck" the square where the agent is resided will changed from "Dirty" to "Clean".

"Agent" class contains simple reflex agent function algorithm to determine its action based upon the input percept. The implementation of Figure 2.8 in the textbook.

"Percept" class represents the state of the environment such as the status of square A (clean or dirty), the status of square B, and the agent location.

Code:

src/main/java/ai/apps/ReflexVaccumDemo.java

```
package ai.apps;

import java.util.ArrayList;
import java.util.List;
import ai.core.Agent;
```

```
import ai.core.Environment;
import ai.core.Percept;

public class ReflexVaccumDemo {

    private static final int totalConfigurations = 8;

    public static void main(String[] args) {
        Agent agent = new Agent();
        List<Percept> percepts = new ArrayList<>(totalConfigurations);
        percepts.add(new Percept(false, false, Percept.Location.A));
        percepts.add(new Percept(false, true , Percept.Location.A));
        percepts.add(new Percept(true , false, Percept.Location.A));
        percepts.add(new Percept(true , true , Percept.Location.A));
        percepts.add(new Percept(false, false, Percept.Location.B));
        percepts.add(new Percept(false, true , Percept.Location.B));
        percepts.add(new Percept(true , false, Percept.Location.B));
        percepts.add(new Percept(true , true , Percept.Location.B));

        PerformanceMeasure performanceMeasure = new PerformanceMeasure(1000);
        double totalScore = 0;

        StringBuilder sb = new StringBuilder();
        for (Percept percept : percepts) {
            sb.append(percept.toString())
              .append(" = ");

            int score = performanceMeasure.evaluate(new Environment(percept), agent);
            totalScore += score;

            sb.append(score).append("\n");
        }
        sb.append("---------------------------\n")
          .append(" Average score = ")
          .append(totalScore/totalConfigurations);

        System.out.println(sb.toString());
    }
}
```

src/main/java/ai/apps/PerformanceMeasure.java

```
package ai.apps;

import ai.core.Agent;
import ai.core.Environment;
import ai.core.Percept;

public class PerformanceMeasure {

    private int timeStep;

    public PerformanceMeasure(int timeStep) {
        this.timeStep = timeStep;
    }

    public int evaluate(Environment env, Agent agent) {
        int score = 0;

        for (int i=0 ; i<timeStep; i++) {
            Agent.Action action = agent.simpleReflex(env.getCurrentPercept());
            env.updateState(action);
            score += computeScore(env.getCurrentPercept());
        }
        return score;
    }

    private int computeScore(Percept percept) {
        int aScore = percept.getAStatus() ? 1 : 0;
        int bScore = percept.getBStatus() ? 1 : 0;
        return aScore + bScore;
    }
}
```

src/main/java/ai/core/Environment.java

```java
package ai.core;

public class Environment {

    private Percept currentPercept;

    public Environment(Percept currentPercept) {
        this.currentPercept = currentPercept;
    }

    public Percept getCurrentPercept() {
        return this.currentPercept;
    }

    public void updateState(Agent.Action action) {
        switch (action) {
            case SUCK:
                currentPercept.clean();
                break;
            case RIGHT:
                currentPercept.setAgentLocation(Percept.Location.B);
                break;
            case LEFT:
                currentPercept.setAgentLocation(Percept.Location.A);
                break;
            default:
                break;
        }
    }
}
```

src/main/java/ai/core/Agent.java

```java
package ai.core;

public class Agent {

    public enum Action { SUCK, RIGHT, LEFT }

    public Action simpleReflex(Percept percept) {
        if (percept.isDirty()) {
            return Action.SUCK;
        } else if (percept.isInA()) {
            return Action.RIGHT;
        } else if (percept.isInB()) {
            return Action.LEFT;
        }
        return null;
    }
}
```

src/main/java/ai/core/Percept.java

```java
package ai.core;

public class Percept {

    public enum Location { A, B }

    private boolean aStatus;
    private boolean bStatus;
    private Location agentLocation;

    public Percept(boolean aStatus, boolean bStatus, Location agentLocation) {
        this.aStatus = aStatus;
        this.bStatus = bStatus;
        this.agentLocation = agentLocation;
    }

    public boolean isInA() {
        return agentLocation == Location.A;
    }
```

```
    public boolean isInB() {
        return agentLocation == Location.B;
    }

    public boolean isDirty() {
        switch (this.getAgentLocation()) {
            case A: return !aStatus;
            case B: return !bStatus;
            default: return false;
        }
    }

    public boolean getAStatus() {
        return aStatus;
    }

    public void setAStatus(boolean aStatus) {
        this.aStatus = aStatus;
    }

    public boolean getBStatus() {
        return bStatus;
    }

    public void setBStatus(boolean bStatus) {
        this.bStatus = bStatus;
    }

    public Location getAgentLocation() {
        return agentLocation;
    }

    public void setAgentLocation(Location agentLocation) {
        this.agentLocation = agentLocation;
    }

    public void clean() {
        this.aStatus = isInA() || this.aStatus;
        this.bStatus = isInB() || this.bStatus;
    }

    /**
     * [ Clean* | Dirty ]
     */
    @Override
    public String toString() {
        StringBuilder sb = new StringBuilder();
        sb.append("[")
          .append(aStatus ? "Clean" : "Dirty").append(isInA() ? "*" : "")
          .append("|")
          .append(bStatus ? "Clean" : "Dirty") .append(isInB() ? "*" : "")
          .append("]");

        return sb.toString(); // [ Clean* | Dirty ]
    }
}
```