Autonomous Agents and Multi-Agent Systems 2020/21
1st Lab: Loading Docks

March 8, 2021

1 Introduction

This document contains the description of the Loading Docks environment and the code provided to implement it. The goal of this lab is for the student to get familiar with the environment and get a hands on work developing a single autonomous agent.

Loading Docks is a frequently used scenario in the field of Multi-Agent Systems. In this scenario, autonomous robots are used to store boxes in corresponding shelves in a warehouse. Boxes are loaded into a central ramp and should be stored according to their color (e.g. blue boxes in blue shelves). Every object in the environment has a regular form and is mapped into grid units. Figure 1 illustrates the initial situation of Loading Docks.

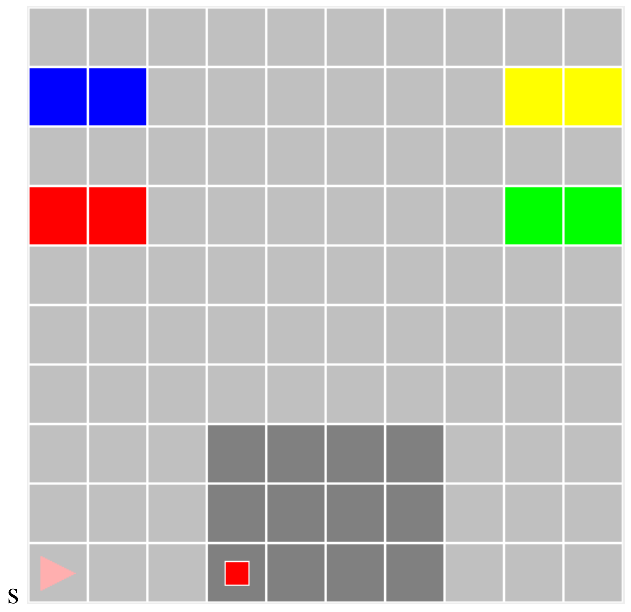

Figure 1: Loading Docks environment.

2 Environment

In this Section, the student is provided a description of the Loading Docks' environment. Please read it carefully before advancing to the next Sections.

2.1 Entities

This scenario has four types of entities:

1. **Robot**: autonomous agent, it can grab a single box at a time, carry and store boxes in a shelf. Additionally, the robot can move one cell forward (if it is free) and rotate in 90º steps (left and right). The robot sensors are limited as it can only sense its current cell and the cell in front of it. It can also sense whether it is carrying a box or not. Further, it can recognize the color of the box it carries.
2. **Ramp**: static object (unable to change its position) that holds boxes. Robots cannot move through the ramp. It initially holds 8 boxes of four different colors. There is only one ramp in the store.
3. **Shelf**: static object characterized by a color where boxes can be stored. The environment has 4 shelves with 4 different colors (red, blue, green and yellow). Each shelf contains two holding compartments; thus, it can hold up to two boxes. It should be noted that,the boxes must store in a shelf with the same color (e.g. red boxes must be stored on the red shelf).
4. **Box**: it is a static object that can be carried by the robots. It is characterized by a color.

2.2 Dynamics

- A robot can move one cell at a time in the direction it is facing or change its direction 90º left or right. It can not move to an occupied cell or go outside the warehouse's limits.
- A robot not carrying a box can grab a box if the box is standing in the cell in front of him. A robot can carry one box at a time and can only drop it in the corresponding shelf. Robots cannot exchange boxes. To store the box in the shelf, the agent needs to be facing an empty shelf of the corresponding color. A robot can only drop a box in the cell ahead, so it should take the right orientation before dropping it.
- Robots' goal is to store all boxes in the corresponding shelves and then return to their initial position. They can only carry one box at a time, cannot exchange boxes or drop them in the warehouse's floor.

3 Setup

The student is provided a Java project. Eclipse IDE for Java is recommended, but please feel free to use the tool that best suits you.

4 Code

You are provided a skeleton code structure that launches the Loading Docks environment. Later you will be asked to implement a single agent. The code provided has a total of 7 files:

- *Agent.java*: This file contains the implementation of a Robot agent. The robot has as properties a position (a *Point* with coordinates *x* and *y*) and the direction it is facing (in degrees). The student will find the following methods to be implemented in this class: *agentDecision*,
- *Block.java*: A block can be a *shelf cell*, *ramp cell* or *free cell*. You do not have to edit this file.
- *Board.java*: This file contains the implementation of the Warehouse. Here you will find the initialization of entities (robots and boxes) and its placement in the defined grid world, as well as the necessary methods for the environment to run. You do not have to edit this file.
- *Box.java*: This file contains the code for the entity *box*. You will need to implement methods for a box to be picked, dropped and moved.
- *Entity.java*: An entity can be a *robot* or a *box*. You do not have to edit this file.
- *GUI.java*: This file contains the backend implementation of the graphical interface that enables the student to see the Loading Docks enviromnent run. You do not have to edit this file.
- *Main.java*: This file contains the main class to be run. You do not have to edit this file.

4.1 Warehouse (*Board.java*)

Warehouse is represented in a 13x13 grid, as shown in Figure 2. The grid cells (patches) may have different colors and each one identifies a different element on the warehouse:

- Dark grey cells: form the ramp where the boxes initially are.
- Blue, red, yellow and green cells:are the blue, red, yellow and green shelves, respectively.
- The remaining cells represent the free floor of the warehouse.

The cells have four relevant properties:

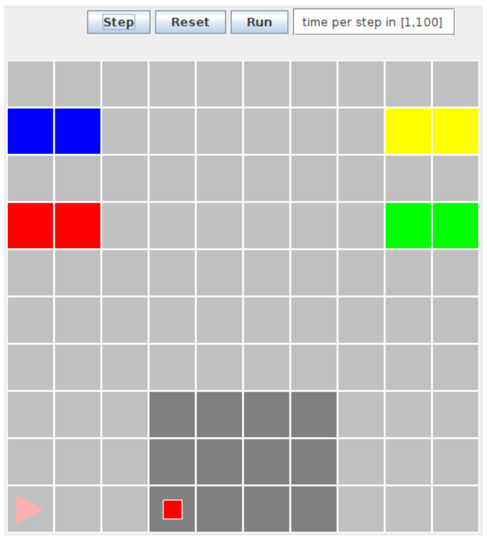

Figure 2: Loading Docks' Java graphical interface.

- shape: defines the type of the cell (shelf, ramp and free). The values of this property might be defined with one of the following variables: *loadingdocks.Block.Shape.shelf*, *loadingdocks.Block.Shape.ramp* and *loadingdocks.Block.Shape.free*.
- color: defines the color of the cell, which only makes sense in case of being a shelf (*shape=loadingdocks.Block.Shape.shelf*), otherwise its value may be ignored. The values of this property might be: *java.awt.Color.blue*, *java.awt.Color.green*, *java.awt.Color.yellow*, *java.awt.Color.red*.

4.2 The Agent (*Agent.java*)

A robot in Loading Docks is defined as an *Entity* and has the following properties:

- point: robot's coordinates in the warehouse grid. It is an instance of class *Point* that contains properties *x* and *y* describing the its position coordinates in the warehouse.
- direction: defines the robot's current direction.

The values of the heading property range between 0 e 360, denoting the clockwise rotation degree of the robot around its vertical axis. The 0 value means the robot is upturned.

Additionally, a robot may also include another property, called *cargo* a *Box* class instance, which contains a reference to the box being carried. When a robot doesn't carry any box, the value of cargo is *null*.

4.3 The Boxes (*Box.java*)

Boxes are also defined as an *Entity* in this scenario. Their main properties are:

- point: define the coordinates in the warehouse grid. It is an instance of class *Point*.
- color: defines the color of the box and might be one of the following variables: *java.awt.Color.blue*, *java.awt.Color.green*, *java.awt.Color.yellow*, *java.awt.Color.red*.

5 Interface

Besides the graphical representation of the grid (Figure 2), the interface of the simulation environment for the Loading Docks scenario also contains 3 Buttons (Figure 3).

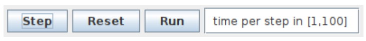

Figure 3: Buttons of the graphical interface for Loading Docks.

The three buttons control the simulation:

- Reset: resets the initial state of the environment;
- Run: continually executes the simulation.By repressing this button, the simulations pauses;
- Step: executes the simulation only one time.

6 Task

Study the Java implementation of the Loading Docks problem using the provided Java project:

- open the Java project in Eclipse and run the Main class;
- explore the provided classes and design the object structure diagram.

After familiarizing yourself with the Loading Docks environment, implement a single agent which goal is to place the red box in the respective dock. For that, the agent needs to:

1. Pick the red box from the ramp;
2. Go to the red dock;
3. Drop the box;
4. Return to the initial position, as well as face its initial direction.

Note that the robot is already able to reach the ramp, but **stays stuck in that position, why?** Please define the sensors and actuators that need to be implemented in the *Agent.java* file to complete this task. The behaviour of the robot is implemented in the *agentDecision* method. Additional methods should also be implemented in *Box.java*, in order to respect the environments' dynamics. Any question that may occur please do not hesitate to ask.