Autonomous Agents and Multi-Agent Systems 2020/21

Lab 3: Deliberative Agents

March 23, 2021

1 Introduction

On this laboratory we are going to develop a deliberative multi-agent system under the BDI framework using the same environment as before: Loading Docks on Fig. 1:

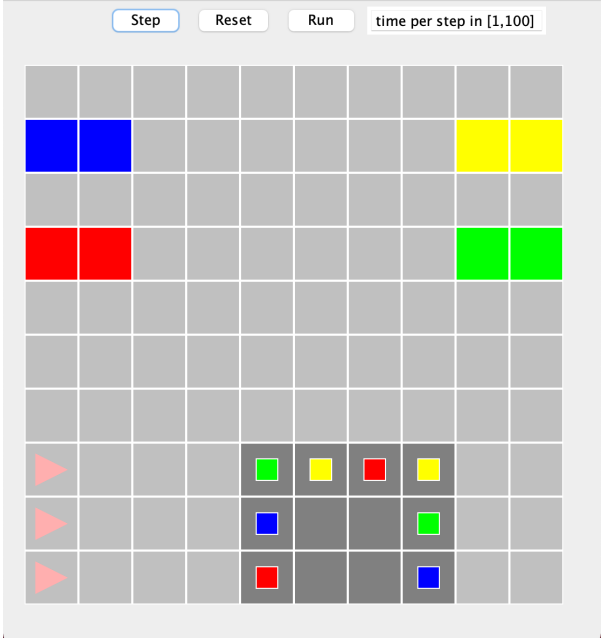

Figure 1: Loading Docks environment.

The section 2 provides context on what was achieved so far – and can be safely be skipped in order to solve today's lab.

2 Background

This section motivates the use of deliberative agents to the loading docks environment and it's aimed at informing by giving a little context on what you have achieved so far. And can be

safely skipped for the task at hand.

2.1 Lab 01

On laboratory 1, you implemented an autonomous agent system in the loading docks environment. The agent had the multi-task goal of (a) navigating to the ramp (b) grabbing a colored box (c) navigating to the shelf (d) dropping the box on the shelf (e) returning home. The agent was a reactive agent with an internal state which informed which sub-task was being carried out on a moment and how to accomplish it.

Furthermore, the environment was fully static – the only changes happening were a direct consequence of the agent's actions. All of those simplifying assumptions let to a hard coded rules approach and the agent was eventually successful.

2.2 Lab 02

On laboratory 2, you saw that the hard coded rules quickly becomes cumbersome as more agents are added on the environment. Which is no longer stationary from the perspective of any single agent: As more then one agent with the same sub-task might reach for the a given square effecting blocking one another. Such state of affairs make it hard to predict all the possible environmental states mitigating the effectiveness of the approach on lab 1.

A different strategy was devised: First, some favorable positions were identified, e.g, if on a ramp facing a box – just grab it or if facing an empty shelf and carrying cargo just drop it. Then randomness was used to solve disputes between agents or break useless patterns as agents kept getting stuck on the top edge. While this approach eventually succeeds it is not effective. Some of you noticed that if the warehouse got larger then the task would become disproportionately harder and some of you developed more rules to prevent agents to do some silly things like passing aside the target ramp.

On this laboratory we will endow agents with deliberative behaviour in order to solve the task more efficiently.

3 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.

3.1 Entities

This scenario is the same as of the previous lab.

3.2 Dynamics

The same dynamics apply, but now the agents should have a deliberative behavior. And now the agents can communicate. Why should the agents communicate? What should the agents communicate? Try to answer those questions before proceeding.

4 Setup

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

5 Task

Agents should implement means-to-reasoning behaviour. The pseudo code seen in the lecture is repeated here for convenience on Fig. 2. The algorithm however seems to be defined for an autonomous agent system instead of a multi-agent system. What are the rules that should be updated for the case of the multi-agent system with communicaton? What lines from algorithm on Fig. 2 should be reviewed? Please try to answer those questions before proceeding. More details on how to implement those methods in the lab please refer to the Section 6:

6 Code

You are provided a skeleton code structure that launches the Loading Docks environment with considerable additions. Including a new broadcasting dynamics and *Agent.java* has gone under extensive additions:

6.1 Broadcasting

- *Board.java*: Has a method with two signatures called `sendMessage` which can be called by the agents to broadcast a message to other agents on a loop.
- *Agent.java*: Has a reference for the game board and may call `sendMessage` directly. Additionally it implements `receiveMessage` to receive a message from all other agents.

6.2 Beliefs-Desires-Intents

Examining the class *Agent.java* we can first locate the cardinal elements for a practical reasoning agent:

- *Desires*: `public enum Desire { grab, drop, initialPosition }`

```
Algorithm: Practical Reasoning Agent Control Loop
1.
2.  B ← B0;        /* B0 are initial beliefs */
3.  I ← I0;        /* I0 are initial intentions */
4.  while true do
5.      get next percept ρ through see(...) function;
6.      B ← brf(B,ρ);
7.      D ← options(B,I);
8.      I ← filter(B,D,I);
9.      π ← plan(B,I,Ac);
10.     while not (empty(π) or succeeded(I,B) or impossible(I,B)) do
11.         α ← head(π);
12.         execute(α);
13.         π ← tail(π);
14.         get next percept ρ through see(...) function;
15.         B ← brf(B,ρ);
16.         if reconsider(I,B) then
17.             D ← options(B,I);
18.             I ← filter(B,D,I);
19.         end-if
20.         if not sound(π,I,B) then
21.             π ← plan(B,I,Ac)
22.         end-if
23.     end-while
24. end-while
```

Figure 2: Practical reasoning control loop

- *Actions*: `public enum Action { moveAhead, rotate, grab, drop, rotateRight, rotateLeft}`
- *Intents*: `public Pair<Desire,Point> intention;`
- *Plan*: `public Queue<Action> plan;`

Where are the beliefs represented? Try to answer this question before moving forward.

6.3 Assignment

Class *Agent.java* has gone to extensive additions. You should implement the following methods in order to endow the agents with the capacity for reasoning: In order for the method `agentDecision()` to be useful you should implement the following methods on the decision code section:

- `buildPlan()`

- `isPlanSound(Action action)`
- `deliberate()`
- `reconsider()`
- `execute(Action action)`
- `impossibleIntention()`
- `succeededIntention()`

You should also implement the communication messages:

- `updateBeliefs()`
- `receiveMessage(Point point, Shape shape, Color color, Boolean free)`
- `receiveMessage(Action action, Point pt)`