B. Nebel, F. Lindner, T. Engesser
Summer Semester 2016

University of Freiburg
Department of Computer Science

Exercise Sheet 3

Due: May 23th, 2016, 10:00

The simulation which we will use in the following exercises was developed by Thomas Bolander, Andreas Garnæs, Martin Holm Jensen, and Mikkel Birkegaard Andersen for an advanced course in planning and multi-agent systems at the Technical University of Denmark. It is inspired by a real-world application, in which robots are used to transport resources (beds, equipment, ...) in a hospital. In the simulation, an agent can only move boxes which match the agent's color. Colors represent different transportation capabilities of the agents. Also, there are different types of boxes. The objective is to have a suitable box delivered to each goal field.

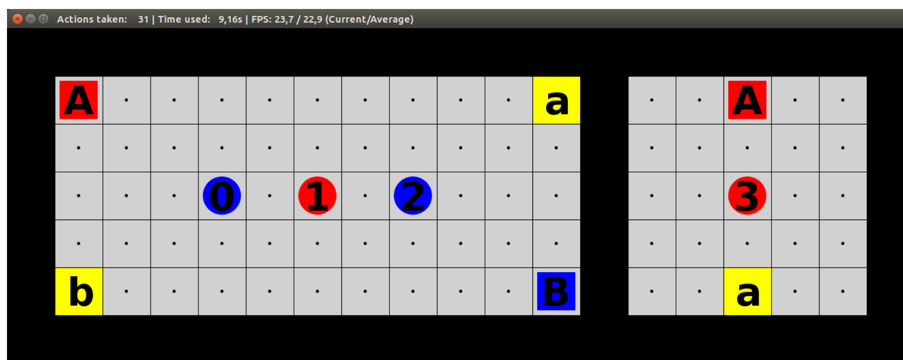

Figure 1: A problem instance. Agents are represented as circles (with a unique id from 0 to 9), boxes and goal fields as colored squares. The type of a box is indicated by an upper case letter, and the box type that is required to be delivered to a goal field by a lower case letter (e.g., a box **A** is required for a goal field **a**).

Since the simulation is written in Java, you will need to have the Java Runtime Environment installed (e.g., in Ubuntu you can do this via `sudo apt-get install openjdk-8-jre`). If the command `java -version` runs in your console, everything should be set up. Agents will be implemented using our multi-agent framework.
Important note: Since communication with the simulation is done via standard in-/output, using `print` statements will break your agents. Instead, use the thread-safe `printout` method provided by the class `SokobanAgent`. Note also that our framework already runs each agent in a dedicated thread. Real-time communication between the agents can be realized via the `SokobanMessaging` actuator.

Exercise 3.1 (Distributed Constraint Satisfaction I, 3+3)

We want to *uniquely* assign each box to *exactly one* agent who can reach the box and has the same color. This means that in the end each agent has exactly one or zero boxes assigned to. If this is not possible for some problem instance, we will consider it unsolvable for now. The related problem of assigning boxes to goal fields will not be relevant for this exercise.

(a) Formalize the problem of box allocation for the instance given in Figure 1 as distributed CSP.

(b) Solve the CSP on paper using asynchronous backtracking. Write down all intermediate steps, including sent messages and changed values of `current_value`, `agent_view`, and `constraint_list`.

Exercise 3.2 (Distributed Constraint Satisfaction II, 6+3)

(a) Implement the allocation of boxes to agents using asynchronous backtracking, starting with the provided template `ex03/sokoban.py`. Note that your agents must first figure out which boxes they can reach. Each message sent during backtracking should also be printed out to the console. Your agents should be able to find allocations for all the levels in `ex03/sokoban-levels/`.

(b) Implement the agents moving to their respective boxes, once they are assigned.