Foundations of Artificial Intelligence

M. Helmert, G. Röger
J. Seipp, S. Sievers
Spring Term 2017

University of Basel
Computer Science

Exercise Sheet 2

Due: March 15, 2017

Exercise 2.1 (1.5+1.5 marks)

Characterize the following environments by describing if they are *static / dynamic*, *deterministic / non-deterministic / stochastic*, *fully / partially / not observable*, *discrete / continuous*, and *single-agent / multi-agent*. Explain your answer.

(a) Minesweeper (the game that comes with many Windows distributions, see, e.g., `https://en.wikipedia.org/wiki/Minesweeper_(video_game)`)

(b) Soccer Robot

Exercise 2.2 (3 marks)

Determine if the following statements about state spaces $\mathcal{S} = \langle S, A, cost, T, s_0, S_\star \rangle$ are correct or not. Explain your answer.
Remark: The cardinality of a set X is denoted by $|X|$.

(a) If all actions have equal costs, each solution for $\mathcal{S}$ is optimal.

(b) From $|S| < \infty$ and $|A| < \infty$ it follows that $|T| < \infty$.

(c) There is no solution for $\mathcal{S}$ if $T = \emptyset$.

(d) If $\pi = \langle \pi_1, \ldots, \pi_n \rangle$ is a minimal cost path from state $s \in S$ to state $s' \in S$ and $\pi' = \langle \pi'_1, \ldots, \pi'_m \rangle$ is a minimal cost path from s' to state $s'' \in S$, then $\pi'' = \langle \pi_1, \ldots, \pi_n, \pi'_1, \ldots, \pi'_m \rangle$ is a minimal cost path from s to s''.

(e) If T is finite, the set of paths in $\mathcal{S}$ is finite as well.

(f) Let π, π', and π'' be minimal cost paths from $s \in S$ to $s' \in S$, s' to $s'' \in S$ and s to s'', respectively. Then $cost(\pi'') \geq cost(\pi) + cost(\pi')$.

Exercise 2.3 (4+2 marks)

The task in this exercise is to write a software program. We expect you to implement your code on your own, without using existing code (such as examples you find online). If you encounter technical problems or have difficulties understanding the task, please let us – the tutor or assistants – know *sufficiently ahead of the due date*. Please also read the *hints* below.
Download the file `state-spaces.tar` from the website of the course. The Java code contains the black box interface for state spaces (`StateSpace`) that was presented in the lecture, and also an interface for actions (`Action`), states (`State`), and a wrapper for state-action pairs (`ActionStatePair`). We also provide an example implementation of the blocks world state space that was presented in the lecture (`BlocksStateSpace`). It implements the given interface. Finally, for running and testing the code, `StateSpaceTest` contains methods that read input files specifying concrete blocks world state spaces in a particular format: The first line specifies the number n of blocks (identified by integers $0, \ldots, n-1$), and the second and third line specify the initial and goal state, respectively, where a state is described by lists of blocks that are stacked above each other (a so called "tower"), separated by -1 to denote different towers. For example, the initial state specified in the file `problem5.bw` has block 4 above block 2 above block 0 on one tower, and block 1 above

block 3 on another one. To run the program, use the following command in a shell (on Linux): `java StateSpaceTest blocks problem5.bw`
Consider a restricted variant of the blocks world state space where blocks are associated with sizes, i.e., in the representation of blocks as integers, block x has size x, which means that block 0 is the smallest block and block $n-1$ is the largest block if there are n blocks. In this restricted variant, it is not allowed to stack a block onto smaller blocks. Furthermore, there is a fixed number of positions on the table that can be used (and hence a fixed number of towers that can be built). Finally, we care about the order of these positions. The latter means that, e.g., the state where the first tower consists of block 0 above block 1 and the second tower is empty is different from the state where the first tower is empty and the second tower consists of block 0 above block 1.

(a) Implement the restricted variant of the blocks world state space, including actions, states and action state pairs, in a file `RestrictedBlocksStateSpace.java`. As for the regular blocks world state space, the parameters of the state space must be specified by the user, which includes the number of blocks n, the number of table positions m (i.e., the number of towers), the initial state and the goal state.

(b) Implement the `buildFromCmdline` function for the restricted variant of the blocks world state space such that it parses an input file that specifies a concrete blocks world state in the following syntax:

- first line: number of blocks n and number of table positions m
- initial state: m lines, where the i-th line contains the IDs of the blocks stacked in the i-th table position, starting from the surface of the table and ending with the delimiter `-1`)
- goal state: analougsly, m lines describing the towers on the table positions as for the initial state.

The file `problem_4_3.rbw` contains the description of a restricted blocks world state space with 4 blocks and 3 table positions. The tower in the first table position is initially empty, the tower in the second table position initially contains the tower of blocks with IDs 3 (table), 1, and 0 (top) and the tower in the third table position contains a tower that consists only of block 2 initially. The goal is to have a tower of blocks with increasing IDs in the second table position.

Modify the method `createStateSpace` in the class `StateSpaceTest` so that if given the keyword `restricted_blocks` (rather than `blocks` as in the example), it calls `buildFromCmdline` of `RestrictedBlocksStateSpace`, i.e., your programm must be callable with the following command: `java StateSpaceTest restricted_blocks problem_4_3.rbw`

Hint: It is sufficient to create one new Java file `RestrictedBlocksStateSpace.java` in the spirit of the example state space implemented in `BlocksStateSpace.java`. In particular, no changes to the other files except `StateSpaceTest` are required!
Hint: Please *test* your solution. Your code must be compilable and we must be able to run it!

Important: The exercise sheets can be submitted in groups of two students. Please provide both student names on the submission. Please create a PDF for exercises 2.1 and 2.2 and a directory containing the Java files for exercise 2.3. Afterwards, please create a zip file containing the PDF and the directory and submit it.