ICS 271A Fall 2021

Lecture A : Introduction to Artificial Intelligence (34750)

Kalev Kask

Project notes : (classic) Sokoban

2021-10-09 : Input/Output format

input is a txt file, containing 5 lines defining the board :

numRows numColumns, e.g. "5 3"

nWallSquares a list of coordinates of wall squares, e.g. "12 1 1 1 2 1 3 2 1 2 3 3 1 3 3 4 1 4 3 5 1 5 2 5 3"

nBoxes a list of coordinates of boxes, e.g. "1 3 2"

nStorageLocations a list of coordinates of storage locations, e.g. "1 4 2"

player's initial location x and y, e.g. "2 2"

All square coordinates (wall, box, storage, player) are : first row index, then column index; e.g. storage at "4 2" means storage in 4^{th} row and 2^{nd} column.

The example above is from : sokoban01.txt and input00.txt

Output is a single line, beginning with nMoves followed by a sequence of letters (U,D,L,R) indicating direction of the move, e.g. "1 D".

There are several sample files :

1. Sokoban__.txt files are the problem spec files in the input format specified above
2. Input__.txt files are corresponding visual-representation files, intended to render the problem in human-friendly format

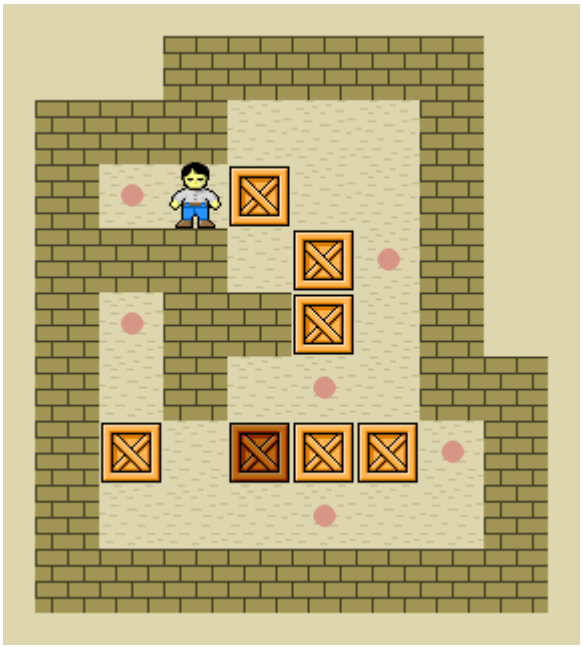
Milestone : Draft Design

Due : 11/21/2021 11:59pm PST

Length : minimum – 4 pages, maximum – 10 pages (excluding title, references, appendix page(s)).

Submission requirements : On the title page, list all team members (names, UCI student IDs). This is a group assignment on canvas. Only one team member needs to submit – reference your original group you submitted to Team Formation Assignment.

Overview : In this assignment, you are asked to submit a software design for the AI (smarts) of the game of Sokoban (https://en.wikipedia.org/wiki/Sokoban).



You need to

1. Design the algorithm(s) and data structures; you don't need to go into detail of standard data structures, e.g. priority queue, heap, hash table, etc.
2. Present the algorithms using pseudo-code (on the same level as algorithms are presented in the textbook). Again, skip standard subroutines, e.g. sorting.
3. Explain/describe the (pseudo-code of the) algorithms.
4. Provide assessment/evaluation of the (expected) time/space complexity of your algorithms.

For the Draft Design you only need to cover the AI part of the system; you can leave out other parts, e.g. input/output, etc.

You should provide sufficient detail of your design so that a person who

1. Is a competent programmer/software developer,
2. Is familiar with Data Structures and Algorithms,
3. Has taken the 271 class and earned an A

can take your Draft Design and faithfully implement it, solely based on your (written) description/explanation, without talking to you.

Minimum AI requirements : In order to successfully complete this project, you have to apply Reinforcement Learning methods and techniques you have learned in this class to provide a design for the Sokoban playing agent that is reasonably intelligent and has, as a matter of principle, a >0 chance of working in practice (on larger than trivial toy boards). For example, your agent is NOT reasonably intelligent if it satisfies the following condition

1. **It uses a straight-forward MDP definition (every board is a state), AND**
2. **It then represents this state space in an explicit, complete table form, AND**
3. **It then runs value/policy iteration (which is a complete deterministic optimal algorithm) on it to find an optimal policy.**

This described design, while fine theoretically, has no chance of working in practice, since the table would never fit in memory and convergence would take a loooong time.

If your design fails the minimum AI requirement, your score will be reduced by 50%.

Criteria for grading :

1. Is the pseudo-code correct/complete (i.e. would we expect it to work)? – 50%
2. Is the description/explanation of the pseudo-code satisfactory? – 50%

Notes :

- Major, critical AI components of your system you refer/discuss should have a formal pseudo-code definition (you should use the same level of detail as pseudo-code in the textbook, as well as an informal description/explanation in text).
- You are well advised to follow a modular design of your software. Try to break it into meaningful self-contained units that perform a particular function and that can be utilized as subroutines from other modules.
- Make sure to declare/describe any important non-trivial data (structures) you use. You can assume that the reader is familiar with basic data structures, e.g. priority queue, hash table, etc. and you can skip the description of these.
- Do you use a heuristic function? If might be a good idea to define it as a separate (pseudo-code) module.
- If you use a sampling-based approach, does it contain a greedy element, a random element? What is the objective function guiding it?
- If you use a sampling-based algorithm, how do you decide when to stop it?