Last updated on Sunday, March 19, 2017

# Sokoban Assignment

*Intelligent Search - Planning*

## Key information

- Submission due at the end of **Week 08** (end of the weekend)
- Submit your work via Blackboard
- Group size: up to 3 people per submission

## Overview

**Sokoban** is a computer puzzle game in which the player pushes boxes around a maze in order to place them in designated locations. It was originally published in 1982 for the Commodore 64 and IBM-PC and has since been implemented in numerous computer platforms and video game consoles.

The screen-shot below shows the GUI provided for the assignment. While Sokoban is just a game, it models a robot moving boxes in a warehouse and as such, it can be treated as an automated planning problem. Sokoban is an interesting problem for the field of artificial intelligence largely due to its difficulty. It has been proven NP-hard. Sokoban is difficult not because of its branching factor of 4 (up, down, left, right), but because of the huge depth of the solutions (many pushes needed!). Additionally, a move may leave the puzzle in a state in which it is impossible to solve it, creating a state of deadlock. For example, a box in a corner cannot be moved out. If that corner is not a goal, then the problem becomes unsolvable.

As the boxes are indistinguishable, there is no difference between pushing one box or any other to a given target. **The player can only push a single box at a time and is unable to pull any box**.

***The aim of this assignment is to design and implement a solver for Sokoban.***

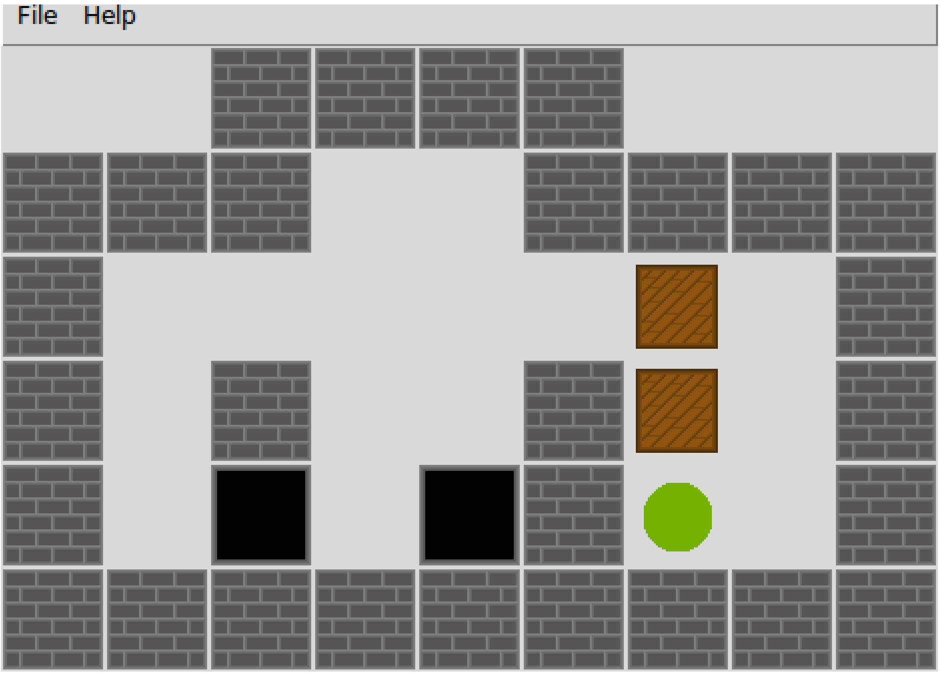

*Illustration 1: Initial state of a warehouse. The green disk represents the agent/robot/player, the brown squares represent the boxes/crates. The black cells denote the target positions for the boxes.*

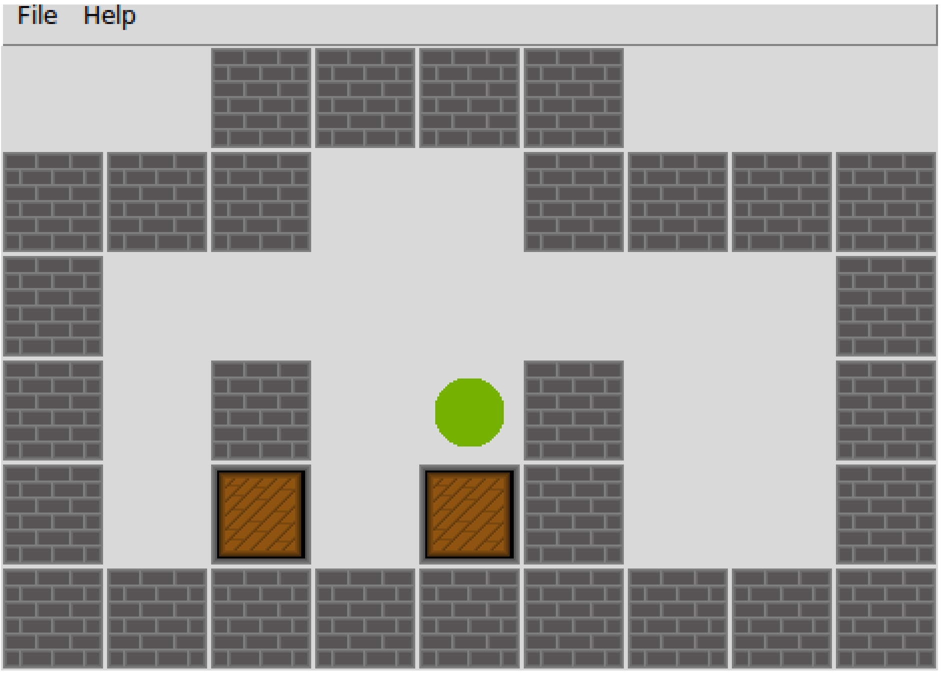

*Illustration 2: Goal state reached: all the boxes have been pushed to a target position.*

### Puzzle representation in text files

To help you design your solver, you are provided with a large number of puzzles.

The puzzles and their initial state are coded as follows,

- **space**, a free square
- ’**#**’, a wall square
- ’**$**’, a box
- ’**.**’, a target square
- ’**@**’, the player
- '**!**', the player on a target square
- '*****', a box on a target square

For example, the puzzle state of the first figure is code in a text file as

```
    # # # #
# # #     # # # #
#           $   #
#   #     # $   #
#   .   . # @   #
# # # # # # # # #
```

## *Suggestions for your search algorithm*

As already observed, Sokoban has a large search space with few goals, located deeply in the tree. Furthermore, lower-bound heuristics can be obtained. These properties suggest to approach the problem with an informed search that finds sparsely distributed goals in a huge search space. Suitable generic algorithms include A* and its variations.

Another useful trick to explore large search space is to use ***macro moves.*** In the context of Sokoban, an ***elementary action*** is the 1 cell move of the worker. A ***macro action*** is the decision of a manager to push one specific box to an adjacent cell. The macro action triggers itself an auxiliary problem; can the worker go next to the specified box. Note that the macro action can (and should!) be translated into a sequence of elementary worker moves.

## *Code provided*

- **search.py** contains a number of search algorithms and search related classes.
- **sokoban.py** contains a class ***Warehouse*** that allows you to read a puzzle from a text file and display its content on the python console.
- **sokoban_gui.py** a GUI implementation of Sokoban that allows you to play and explore puzzles. This GUI program does not solve puzzles, it simply allows you to play!
- **mySokobanSolver.py** code skeleton for your solution. You should complete all the functions located in this file.
- **tester_script.py** script to perform basic tests on your solution.
- A large number of puzzles in the folder 'warehouses'

## *Your tasks*

Your solution **has to** fit in the same framework as the one used in the practicals. That is, you have to use the classes and functions provided in the file **search.py**.

All your assessable code should be located in the file called **mySokobanSolver.py. This is the only Python file that you should submit**. In this file, you will find partially completed functions and their specifications. When your submission is tested, it will be run in a directory containing the files *search.py* and *sokoban.py*.

## *Deliverables*

You should submit via Blackboard a zip file containing

1. A report in **pdf** format **strictly limited to 4 pages** (be concise!)
   - explain clearly your heuristics, and other important features of your solver
   - describe the performance and limitations of your solver
   - use tables and figures
2. Your Python file **mySokobanSolver.py**

## *Marking Guide*

- **Report**: 5 marks
  - Structure (sections, page numbers), grammar, no typos.
  - Clarity of explanations.
  - Figures and tables (use for explanations and to report performance).
- **Code quality**: 5 marks
  - Readability, meaningful variable names.
  - Proper use of Python constructs like dictionaries and list comprehension.
  - Header comments in classes and functions.
  - Function parameter documentation.
  - In-line comments.

- **Functions of mySokobanSolver.py**
  - **my_team():** 1 mark
  - **taboo_cells()**: 3 marks
  - **check_action_seq()**: 3 marks
  - **solve_sokoban_elem()**: 4 marks
  - **can_go_there()**: 3 marks
  - **solve_sokoban_macro()**: 6 marks