Sokoban AI Agent

IFN680: Artificial Intelligence and Machine Learning
Project 2

1 Key Information

- Submission due at the end of **Week 13** (27 October, 23.59pm).
- You will be working in groups of 2 for this project.

2 Overview

Sokoban is a computer puzzle game in which the robot 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 screenshot below shows the Sokoban environment 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 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 bad move may leave the puzzle in a doomed state from 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 an AI agent for the robot worker capable of solving Sokoban.

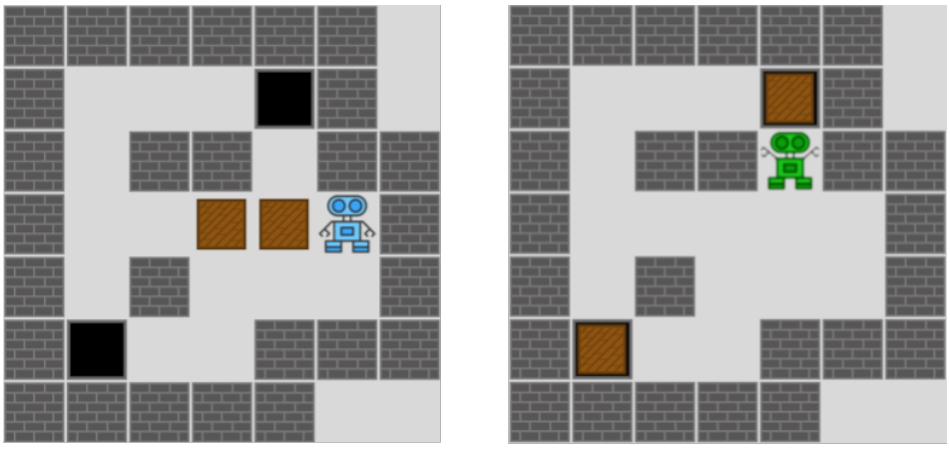

Figure 1: Sokoban environment illustrating the robot, boxes and goal positions for the boxes.

2.1 Environment representation in text files

To help you design your solver, you are provided with a large number of different environments. The environment and their initial states are coded as follows:

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

For example, the initial environment shown in Figure 1 can be coded as a text file as follows:

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

3 Task

Graph search algorithms play a crucial role in the field of Artificial Intelligence, especially in the context of problem-solving and decision-making in various applications. One of these applications includes games and puzzles, where these algorithms are used to traverse the game space and find optimal solutions.

In this assignment, you will apply your understanding of graph search algorithms in order to solve the Sokoban game. Sokoban is a compelling representation of a graph problem, with each state of the game representing a node and each valid move forming an edge. Your task is to leverage graph search algorithms taught in this unit to navigate through the game state space and find the **optimal** path to the goal state.

A key component of this task will be to design an algorithm that is capable of solving the Sokoban game in the most efficient manner possible. Given that the Sokoban environments can exhibit very

large search spaces a key task to increase the efficiency of your search, involves optimizing the size of this search space where possible. One effective method of doing so is identifying and marking **taboo cells** - possible box locations that, once pushed into, would make the game unsolvable. By recognizing these cells, your algorithm can avoid searching these paths and thus reduce the search space significantly.

Another useful tool to explore large search space is to use **macro moves.** In the context of Sokoban, an **elementary action** is the one-step move of the worker. A **macro action** is the decision of a manager to have one specific box pushed to an adjacent cell. The macro action triggers itself an auxiliary problem; can the worker go to the cell next to the specified box. Note that the macro action can be translated into a sequence of elementary worker moves.

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`.

3.1 Code Provided

- `search.py` contains a number of search algorithms and search related classes.
- `sokoban.py` contains a class Warehouse that allows you to load a puzzle from a text file and display its content on the python console.
- `mySokobanSolver.ipynb` contains the skeleton code for your solution. You should complete all the functions located in this file.
- `marking_script.ipynb` script to perform basic tests on your solution.
- A large number of puzzles in the folder 'warehouses'
- `sokoban_gui` you will be provided a download link for 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 and gain an understanding of the game. Download both the warehouses folder and the appropriate GUI for your operating system (see Assignment 2 page on Canvas).

All your code should be located in a single 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`. If you break this interface, your code will fail the tests.

3.2 Deliverables

You should submit via Canvas a zip file containing:

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

4 Marking Guide

- Report: **10 marks**
  - Structure (sections, page numbers), grammar, no typos.
  - Clarity of explanations.
  - Figures and tables (use for explanations and to report performance).
  - A statement on each team member's contribution to the project.
- AI Solver Implementation in `mySokobanSolver.ipynb`: **20 marks**
  - `my_team()`: **1 mark**
  - `taboo_cells()`: **3 marks**
  - `check_action_seq()`: **3 marks**
  - `solve_sokoban_elem()`: **3 marks** + **1 mark** for solver optimality
  - `can_go_there()`: **3 marks**
  - `solve_sokoban_macro()`: **3 marks** + **1 mark** for solver optimality

  Code Quality: **2 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.

  Each function has to produce the correct output in order to gain the mark indicated for the corresponding function. For both `solve_sokoban_elem()` and `solve_sokoban_macro()` an additional 2 marks are given if your solver can produce a solution with an optimal number of steps. Here we are looking at your ability to leverage techniques beyond simple uninformed search.

  A detailed marking rubric for this assignment is provided at the end of this task sheet.

5 Final Remarks

- Do not underestimate the workload. Start early. You are strongly encouraged to ask questions during the workshop sessions.
- Email questions to `ranak@qut.edu.au`
- Enjoy the assignment!

Criteria	Ratings				Pts
Code Submission					**20 Pts**
Solver Function Evaluations.	The functions are evaluated using a marker script. The script checks if each function can produce the correct outputs when given a specific warehouse in the test set. A function is provided with the full set of marks if it runs without errors and produces the expected output. **The breakdown of marks for each function is given in the task sheet.**				**16 Pts**
Optimality of the solver	**2 Pts** Both the elementary and macro solvers achieve a number of steps that are less than the output from an uninformed search algorithm.	**1 Pts** At least one of the solvers achieves a number of steps that is less than the output from an uninformed search algorithm	**0 Pts** The solver is equivalent to the performance of an uninformed search algorithm,	**-**	**2 Pts**
Code Quality. This criterion measures the readability of your code, meaningful variable names, the proper use of Python constructs, header comments and inline comments to aid the clarity of your code.	**2 Pts** The code is modular and well-structured. It runs without any errors. Proper use of data structures. Header comments are clear. The new functions can be unambiguously implemented by simply looking at the header comments. Each function parameter is documented (including the type and shape of the parameter). Useful in-line comments and thoughtful choice of variable names. Appropriate use of auxiliary functions.	**1 Pts** The code is modular and well-structured. It runs without any errors. Header comments are clear. Useful in-line comments to explain what the student attempted to implement	**0 Pts** The code is difficult to follow and is plagued with errors. No useful header or inline comments were provided to explain the code.	-	**2 Pts**
Report Submission					**10 Pts**
Grammar and Clarity of Presentation	**2 Pts** The report is written at the highest professional standard with respect to spelling, grammar, formatting, structure, and language terminology.	**1 Pts** The report is generally well-written and understandable but with a few minor presentation errors that make one or two points unclear.	**0 Pts** A large part of the report is poorly written, making many parts challenging to understand.	-	**2 Pts**
Discussion and analysis of the proposed solution.	**5 Pts** The implemented solution and justification are clearly presented. Clear explanations of the heuristics used. Clear figures or tables illustrate comparisons conducted.	**3 Pts** The report contains some explanation of the implemented code. An explanation of the heuristics used is provided. No figures or tables were provided.	**1 Pts** An attempt is made to explain the implemented code. No discussion of the heuristics used. No figures or tables were provided.	**0 Pts** There is no evidence to explain or discuss the implemented solver.	**5 Pts**
Discussion of limitations.	**3 Pts** The limitations of the proposed method are clearly explained and well justified. Future work recommendations are provided	**2 Pts** The limitations of the proposed method are clearly explained and well justified.	**1 Pts** An attempt is made to list a limitation of the method.	**0 Pts** There is no evidence to explain or discuss the limitations of the proposed method.	**3 Pts**
Total					**/30 Pts**