Major Course Output 1
SokoBot

Release Date: September 15, 2023
Due Date: October 22, 2023 (Sun), 11:59 PM

1 Project Description

In this project, students will apply the concepts and theories they have learned in the course to develop an artificial intelligence algorithm for a bot that plays a classic puzzle game. Through this project, students should be able to demonstrate the following learning outcomes:

- **LO1.** Design and evaluate informed search algorithms and knowledge representations for problem solving.
- **LO2.** Collaboratively build systems that consider a number of paths or strategies in order to improve its performance in achieving its goal in less amount of computing time, or by some other metric of performance.
- **LO5.** Articulate ideas and present results in correct technical written and oral English.

2 Project Specifications

This section contains the specifications for this major course output.

2.1 Overview

In this project, you will design and implement an artificial intelligence algorithm for a bot that plays Sokoban. You must represent the game as a state-based model and apply the necessary techniques and algorithms to automatically solve puzzles in a reasonable time frame.

2.2 Sokoban

Sokoban is a classic Japanese puzzle game. It was created by Horiyuki Imabayashi and was first published by Thinking Rabbit in 1981. The name Sokoban (倉庫番) translates to "warehouse keeper". The game has been ported to numerous platforms and has inspired countless unofficial clones.

Sokoban is a grid-based puzzle game where the player organizes crates in a warehouse. The rules of the game are as follows:

1. The player can move in the four orthogonal directions (up, down, left, and right).
2. The player can only stand on empty spaces or target spaces. The player cannot walk through walls.
3. The player can push a crate, only if there is an empty space behind it. The player cannot push multiple crates at once.
4. The player cannot pull crates.
5. The goal of each level is to put all crates on the target spaces.

It is recommended that you try to play a few levels of Sokoban first to get a clear understanding of its rules. A free online implementation of the game can be found in this link.

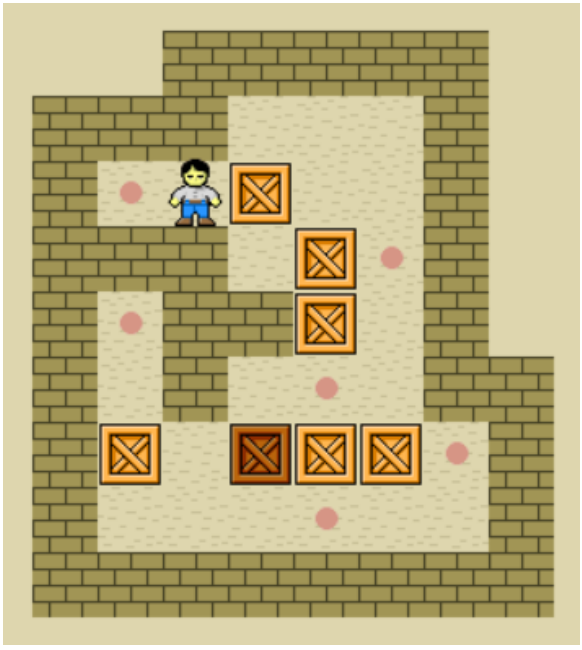

Figure 1: A Sokoban puzzle (click here to see an animated solution for this puzzle)

Despite its simplicity, levels of Sokoban can be very difficult. The difficulty comes from its large branching factor and the fact that solutions in more complex levels can be very deep. Skilled Sokoban players are good in discarding futile and redundant lines of play through the recognition of patterns and subgoals.

Sokoban is a well-studied problem in the field of computer science, as it has implications for real-life applications like motion-planning in robotics. In this project, you will design and implement an algorithm for automatically solving easy Sokoban puzzles.

2.3 Starter Program

You are provided a starter program for this project written in Java. Before anything else, you must first test whether the program compiles and runs successfully on your machine.

Note: For this project, you need a Java compiler and runtime environment installed on your machine. If you do not have this yet, there are many tutorials online on how to install JDK (Java Development Kit). Before you proceed, the `javac` and `java` command should be available from the command line.

There are two modes in this program: **Free Play Mode** and **SokoBot Mode**. In Free Play mode, you can play Sokoban by yourself using the arrow keys. In SokoBot Mode, the program runs a function to generate a solution and plays an animation of that solution on the screen. To run the program, you need to recompile and run the Java source files. For Windows, you are provided two batch files `freeplay.bat` and `sokobot.bat` to do this easily. You can open the batch files in a text editor to see what commands they use.

First, try to run Free Play mode by opening the command line, going into the directory of the project, and invoking the batch file as follows:

```
freeplay testlevel
```

Here, `testlevel` is the name of the level to be loaded. You can replace this with a different level name later if you want. The levels are stored inside the `maps` directory. If successful, you should see a window displaying the level. From here, you can play the game by pressing the arrow keys.

Next, try to run SokoBot Mode using the following command:

```
sokobot testlevel
```

Once the window loads, you can press Space to start the bot. In the starter program, the bot is completely stupid. It simply thinks (sleeps) for 3 seconds, and then simply moves left and right repeatedly for a number of times. Your task for the project is to rewrite the behavior of this bot to make it intelligent.

If you encounter any problems in running the program on your machine, please consult the instructor for guidance.

2.4 Levels and Level Format

Levels should be stored as `.txt` files inside the `maps` directory. The levels should follow the format specified in this link, without any run-length encoding. Additionally, each level should be in its own separate text file. You don't have to worry about writing code to read the files, as the program already does this. Nonetheless, you may want to be familiar with the format if you want to create your own levels for testing. Additionally, you can also download thousands of levels online from various sources.

There are many sample levels provided to you along with the starter program.

2.5 Designing and Implementing SokoBot

Your main task for this project is to design and implement SokoBot, which automatically solves a Sokoban puzzle. To do this, you will change the implementation of the `solveSokobanPuzzle` method in the `SokoBot` class inside the `solver` package.

The `solveSokobanPuzzle` accepts four parameters: `width`, `height`, `mapData` and `itemsData`.

`width` and `height` are integers representing the width and the height of the level.

`mapData` and `itemsData` are both two-dimensional `char` arrays representing the layout of the level. `mapData` contains the static elements of the level, while `itemsData` contains the elements that can move around. `mapData` contains `'#'` for a wall, or `'.'` for a target. On the other hand, `itemsData` contains `'@'` for the player, or `'$'` for a crate. All other elements in `mapData` and `itemsData` are set to space characters (`' '`). Note that it is possible for crates and the player to be in the same location as the targets.

As an example, the level shown in the image above is represented with a `width` of 7, and a `height` of 5, with `mapData` `itemsData` set up as follows:

```
mapData =   {{' ', '#', '#', '#', '#', '#', '#'},
             {' ', '#', ' ', ' ', '.', ' ', '#'},
             {'#', '#', ' ', ' ', ' ', ' ', '#'},
             {'#', '.', '.', ' ', '.', ' ', '#'},
             {'#', '#', '#', '#', '#', '#', '#'}}
```

```
itemsData = {{' ', ' ', ' ', ' ', ' ', ' ', ' '},
             {' ', ' ', '@', ' ', ' ', ' ', ' '},
             {' ', ' ', '$', '$', '$', ' ', ' '},
             {' ', ' ', ' ', ' ', '$', ' ', ' '},
             {' ', ' ', ' ', ' ', ' ', ' ', ' '}}
```

The `solveSokobanPuzzle` method should return a string representing the bot's solution to the given puzzle. The solution string contains the list of moves to be done by the bot, in order. The four directions (i.e., four possible moves) are represented by `'u'`, `'d'`, `'l'`, and `'r'` for up, down, left, and right respectively.

You are allowed to create additional variables, methods, and classes inside the `solver` package. However, you are **NOT** allowed to modify any other codes in the other packages. You are allowed (though not required) to use external libraries for data structures such as trees and graphs, but you are **NOT** allowed to use libraries that perform any artificial intelligence algorithms like search.

In addition to solving the puzzle, your bot should also return a solution in a timely manner. To be more specific, the program imposes a **15-second thinking time limit** for the bot. If your method does not return a solution within that time frame, the program will refuse to display your solution and will simply display a message that the SokoBot took too long thinking. You must take this time limit into consideration in coming up with your algorithm.

In order to come up with an effective bot in this project, you may need to combine and tweak the algorithms and concepts discussed in class. Simply applying the algorithms naively will result in a poorly-performing bot. Thus, you are encouraged to be creative, critical thinkers throughout the process of this project.

2.6 Evaluation

You are expected to do an evaluation of your bot by testing it on different Sokoban levels. You are encouraged to test it not only on the provided sample cases, but also on other levels as well. As you test the bot, focus not only on whether it was able to solve the levels or not, but also on other aspects such as the number of moves it took to solve the puzzle, the time it took for the solution to be found, among others.

2.6.1 Performance Expectations (Grading)

Sokoban has been proven mathematically to be an NP-Hard problem. This means that there is currently no known algorithm that efficiently solves all possible Sokoban levels. However, simpler levels should be solvable. In this project, your bot is not expected to be capable of solving the most complex Sokoban levels, but it must be able to handle as many levels as it can. Your bot will be evaluated on a set of levels (this is hidden to you) with varying difficulty, split across different categories. To get full credit, your bot needs to solve at least 50% of the levels that will be used for testing.

Category	Ratio of Test Levels	Description
A	30%	2 crates
B	20%	3 crates
C	20%	4 crates
D	15%	5 crates
E	15%	more than 5 crates

3 Report

In addition to the SokoBot codes, you are required to write a report documenting the implementation of the project. At the minimum the report should contain the following:

1. **Brief Introduction**: A short introduction describing the project. Please make this section short and concise.
2. **SokoBot Algorithm**: A section describing in detail the algorithm that was used for the implementation of the bot. Focus more on the project-specific details (e.g., how a state was represented, how the actions were generated, the rationale behind the implementation of certain search strategies) rather than the descriptions of already-known concepts.
3. **Evaluation and Performance**: A section describing the performance of the bot as determined through your own evaluation. This section must highlight the strenghts and weaknesses of the bot. Try to cite specific examples where the bot performs well / not well.
4. **Challenges**: A discussion on the difficulties encountered when developing the bot. What are the challenges that make it difficult for automated solvers to tackle the more complex Sokoban problems?
5. **Conclusion:** A conclusion section summarizing the process of implementing the project, including reflections, realizations, and insights obtained.
6. **Table of Contributions:** A table showing the contributions of each member to the project.

There is no required format for the report, but please refrain from submitting very long reports with little substance. Please minimize copying or paraphrasing already known concepts and theories to make the report longer. The bulk of the report should contain the group's personal ideas and insights.

4 Deliverables

There are two deliverables for this project: a zip file containing the source files for the project, and a pdf file containing the report. The zip file should include all the source code and files needed to run the project (include the whole project directory).

The file name of the zip file should be: `mco1_<surname 1>_<surname 2>_<surname 3>_<surname 4>.zip` with the surnames in alphabetical order. The file name of the report should be `mco1_<surname 1>_<surname 2>_<surname 3>_<surname 4>.pdf` with the surnames in alphabetical order.

5 Academic Honesty

Honesty policy applies. Please take note that you are **NOT** allowed to borrow and/or copy-and-paste in full or in part any existing related program code from the internet or other sources (such as printed materials like books, or source codes by other people that are not online). **Violating this policy is a serious offense in De La Salle University and will result in a grade of 0.0 for the whole course**.

Please remember that the point of this project is for all members to learn something and increase their appreciation of the concepts covered in class. Each member is expected to be able to explain the different aspects of their submitted work, whether part of their contributions or not. **Failure to do this will be interpreted as a failure of the learning goals, and will result in a grade of 0 for that member**.