SokoBot – AI Sokoban Solver

Major Course Output 1 (MCO1) CSC613M

I. Introduction

Sokoban or *Warehouse Keeper* is a classic Japanese puzzle game wherein players are tasked to push the boxes or crates to their respective destinations. Aside from the main objective of the game, other mechanics and restrictions will become clear and to any individual playing even for the first time. In this course, the class is tasked to create a bot that can play on any category as seen on *Table 1*.

Category	Description
A	2 crates
B	3 crates
C	4 crates
D	5 crates
E	More than 5 crates

Table 1. Level categories

For this course output, our group decided to focus on the Breadth-First Search Algorithm or BFS for an easier and complete implementation. Though any informed search method may be better with the presence of heuristic values compared to any blind search method. However, we worried that it might not be completed due to time constraints.

II. SokoBot Algorithm

The source code for this project was mostly done in Python, again for easier implementation on the group's part. It was then connected to 'Sokobot.java' by creating a HTTP client connecting via a RESTful service call to a Python service that acts as the http server.

Element	Representation
#	Wall
.	Goal
@	Player
$	Box/Crate

Table 2. Layout elements

As seen on *Table* 2, every state or node in the search tree is represented by arrays with the corresponding layouts for static elements - Wall ('#') and Goal ('.'), and moveable elements - Player ('@') and Box/Crate ('$') respectively. We define first the initial Parent State mainly consisting of the following: 'mapData' array, 'itemsData' array, and player position [x][y]. Succeeding states - Child State(s) originating from the Parent State are defined after the bot takes a valid move through the method move_player(). The Children States will return updated values in the 'mapData' and 'itemsData' arrays. Possible states illustrated below in Figure 1.

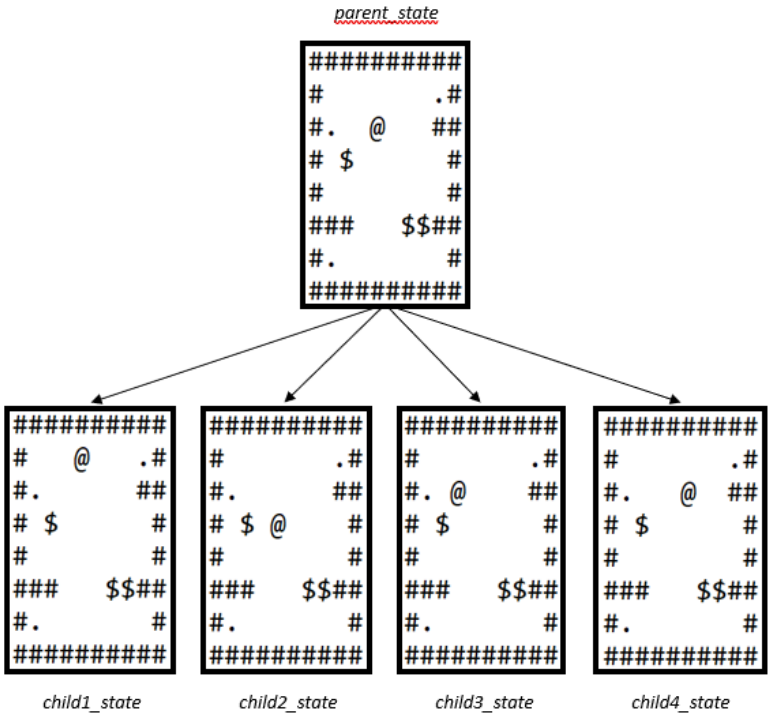


Figure 1. Parent-Children State

The player can only move/push boxes upward(‘u’), downward(‘d’), leftward(‘l’), and rightward(‘r’). Each move is checked through the function is_valid(), whether the player can move in the respective direction, if the box can be pushed, and if there are any errors with updated mapData and itemsData.

The goal state – is_goal() is then defined if the indices of the boxes/crates occupy the same indices as the goal elements. Upon completion by the bot, all valid moves from the respective child states will be appended after on another through the function get_valid_moves() and returned as a string e.g. “*rrrddlllrrruulll..*”

III. Evaluation and Performance

The BFS algorithm was tested through different categories as seen on *Table 1*. Besides the resulting string printed Output generated on sample test levels below are: ‘Generated’ - represents the number of states/nodes created throughout the search, ‘Repeated’ - the number of states explored more than once, ‘Fringe’ - or *Frontier* is the number of nodes that are yet to be explored, and ‘Explored’ - is simply the number of states explored by the search algorithm.

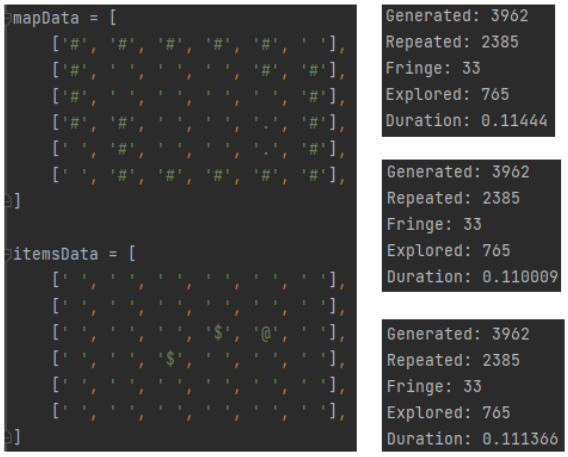
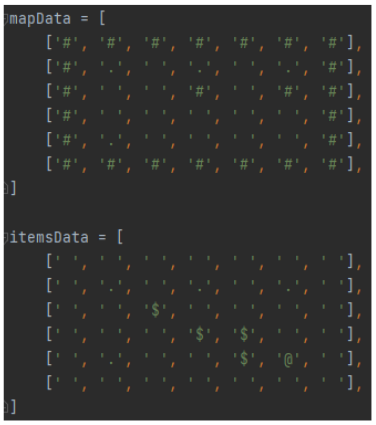


Figure 2. Category A test level



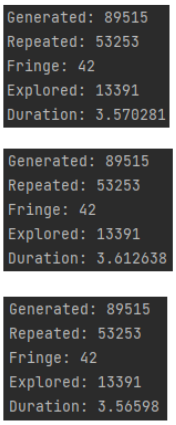


Figure 3. Category C test level

In Fig. 2 and 3, the BFS algorithm yielded favorable results having average runtimes of 0.111938 seconds and 3.582966 seconds for categories A and C respectively. A test run on a category D level was done but decidedly to terminate it as it continuously ran for almost an hour. There was no test done on a category E level as we deduced that the bot was limited to performing only until category C levels due to the 15-second thinking time limit.

IV. Challenges

Challenges that may give automated solvers on this game are complex levels as experienced by using the BFS search algorithm, conflicts and function call errors overlooked between the source codes.

On the group's part, there are a few factors that proved to have major impact during the development stage. Firstly, the required output for this project is in Java language and the group did not have ample familiarity with Java. Secondly, a connection to Java was needed because the code was predominantly done in Python as discussed in *Section II* resulting in more steps on the development. Lastly, was time. Since this is graduate studies, all members have different arrangements outside of work and class schedules.

V. Conclusion and Insights

Upon the implementation of the Breadth-First Search algorithm on the Sokoban game, we learned that the bot had significant limitations discussed in *Section III*. The BFS algorithm can't efficiently solve a level with 5 boxes and more. We can take this as a great opportunity to improve upon and incorporate various search algorithms. On the other hand, on simpler levels with 4 boxes and less, the bot solves the game coherently well within the 15-second thinking limit.

Finally, the BFS search method takes up a lot of time and memory since it explores each state by level. Moving forward, for a more comprehensive report, comparison between the different search algorithms proves to be best. With the presence of sound heuristic values, Greedy Best First Search (GBFS) and A* Search will yield the fastest and most efficient solution respectively (Li, et al., 2019).

VI. Table of Contributions

Name	**ID No.**	**Contribution**
Calamiong, Yno Andrei	12383805	-Python source code -Python to Java conversion -Test/debugging -Review of related research
Hontiveros, Jan Aldo	12383783	-Written report -Review of related research
Mantuano, Joseph Paulo	12385859	-Python to Java conversion -Test/debugging -Review of related research

References

[1] H. Gao, X. Huang, S. Liu, G. Wang, and Z. Zhong (2017).
A Sokoban Solver Using Multiple Search Algorithms and Q-Learning
Hong Kong University of Science and Technology
https://github.com/XUHUAKing/sokoban-qlearning/blob/master/sokoban-solver-final-report.pdf

[2] L. Zhenhan, Z. Yue, and J. Yang (2019).
AI Sokoban Solver: CS271 Project Report
University of California, Irvine
https://github.com/dabaitudiu/SokobanAI/blob/master/SokobanAI.pdf