SokoBot : An Automated Sokoban Solver

Ethan Axl S. Burayag, Ezra Jeonadab G. Del Rosario, Elisha Jeremy C. Ong
De La Salle University - Manila

Introduction : Sokoban and SokoBot

Sokoban is a classic Japanese puzzle video game designed and developed by Hiroyuki Imabayashi in 1981 and published by Thinking Rabbit in 1982. This game revolves around the main character, a warehouse worker, or “Sokoban” in Japanese, who aims to move and push boxes onto different targets or storage locations on a warehouse-like environment. This game is simple yet challenging due to the different rules and limited movement options present in the game. Nonetheless, this puzzle game requires in-depth strategy and problem-solving skills to solve each puzzle efficiently.

SokoBot is an artificial intelligence system developed by developers that automatically solves a range of *Sokoban* puzzles. This system was designed by applying various concepts and algorithms presented in the CSINTSY course. Mainly, this system leverages state-based modeling and incorporates various search algorithms, such as Breadth-First Search (BFS) and A* Search, in order for *SokoBo*t to efficiently explore possible moves, avoid deadlocks, and find an optimal solution to solve various puzzles. Generally, by applying various concepts and strategies, *SokoBot* intelligently and efficiently strategizes and searches for optimal solutions for various puzzles.

SokoBot Algorithm

The SokoBot uses an A* informed search algorithm to efficiently solve Sokoban problems by expanding nodes based on the node with the lowest sum of incurred and heuristic costs. The environment is represented as positions or sets of positions using the Position class marking walls, targets, crates, and the player. Sets are used to avoid redundancy in grouped positions. A State class stores dynamic entities like the player and crate positions. Moreover, SearchNode class represents each node in the search tree, containing the previous node, state, action, and cost. Possible actions are generated by filtering out invalid moves, such as those colliding with walls or causing deadlocks. Nodes are added to the frontier based on A* evaluation, where the cost of each move is determined based on whether or not it adversely affects the evaluation of the goal state, and the heuristic uses Manhattan distance to calculate the player-crate and crate-target distances. This approach allows the SokoBot to solve the puzzle with minimal moves as efficiently as it can within a time constraint. For a more detailed sequence of the algorithm, refer to the flowchart provided in figure 1.

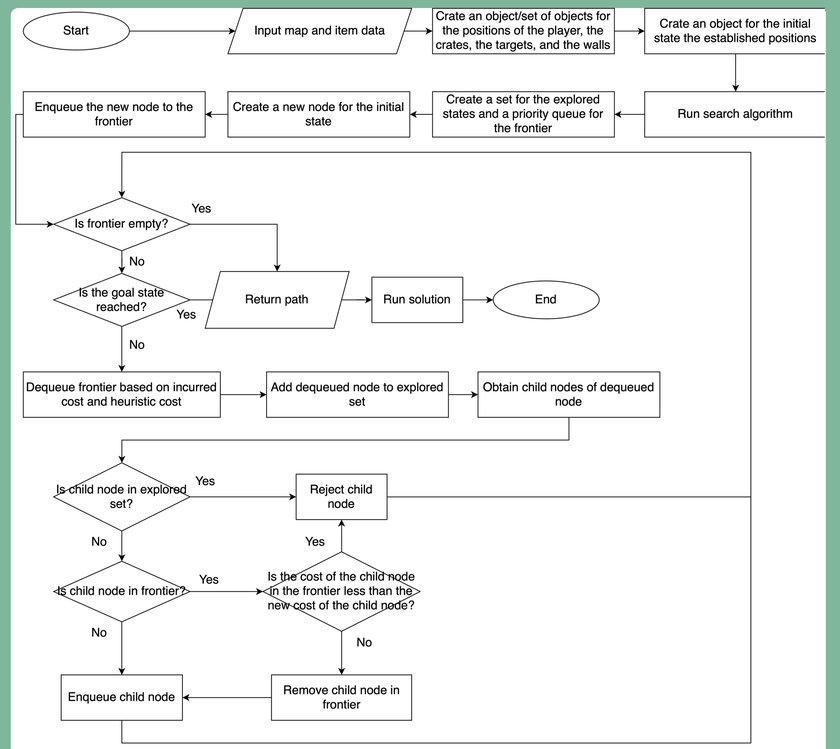

Figure 1. SokoBot Algorithm Flowchart

Evaluation and Performance

The SokoBot was tested with the 16 test maps provided by the professor, with it being run with the recommended 15 second thinking time given some exceptions. It was run three times, getting multiple sample times per map, and the number of moves made by the bot was counted per run.

The A* algorithm utilized showed successful clears for every map except original2 and original3, successfully solving each map in less than 15 seconds, albeit with non optimal moves as some solutions found by the A* algorithm had up to double the moves of the solution found by the original BFS algorithm. The difference in moves was caused by the way the heuristic values were set, which will be expounded later in the challenges section. On the plus side, the A* algorithm was significantly faster, running up to 100x faster than the original BFS algorithm on maps with a small number of boxes, while being able to clear a majority of the maps that timed out with the BFS algorithm.

Overall, the A* checked off all the boxes it was given, successfully clearing most maps within the time limit. A possible optimization that could be done to balance the speed and optimality would be the iterative deepening A* search algorithm, which has similar exploration with the iterative deepening of the DFS-ID algorithm, but with the pathing of the given A* algorithm. Other heuristics may also be set to balance the moves out, but at a cost of more time, and other pruning techniques to cut out obvious edge cases may be done as well. The code performed to the expectations given, and further optimization was deemed unnecessary.

Challenges

The main difficulty revolved around making the SokoBot “optimally smart.” The base A* algorithm became the foundation for the SokoBot. Although with adequate time and resources it can solve most problems, difficulties arise with influencing to make smarter decisions within a time constraint. Developers focused on refining the heuristic evaluation to improve the bot's decision-making, prioritizing states that bring it closer to the goal. However, this refinement introduced issues like backtracking and unnecessary moves, as the bot sometimes made non-progressive moves. To address this, developers implemented cost penalties and rewards to help the bot make more purposeful decisions and avoid actions that lead away from the goal.

Conclusion

The experiences that the developers have gone through with the implementation of SokoBot made them realize the challenge and difficulty of implementing an artificial intelligence system but as well as the satisfaction and fulfillment at the end. Particularly, the implementation of this system took a long time of analysis and implementation before the final product was obtained. From game knowledge representation, base algorithm formulation and implementation, to algorithm testing and optimization, all of these processes took significant amounts of time and effort to come up with a refined final product. Significantly, this entire process made them realize the importance of having a creative and open yet critical and enduring mind every step of the way to ensure an efficient and complete product at the end. Generally, this entire process may be challenging and exhausting, but having the right mind to approach and implement this system will lead to a better result.