Sokoban Game Mini-project Report

Objective :

The main objective of this mini-project was to apply A\* algorithm in implementing Sokoban game solver. This mini-project was done towards the completion of a part of the Artificial Intelligence(1DL010) course under the guidance of  Andreina Francisco and supervised by Jingwei Hu.

Introduction :

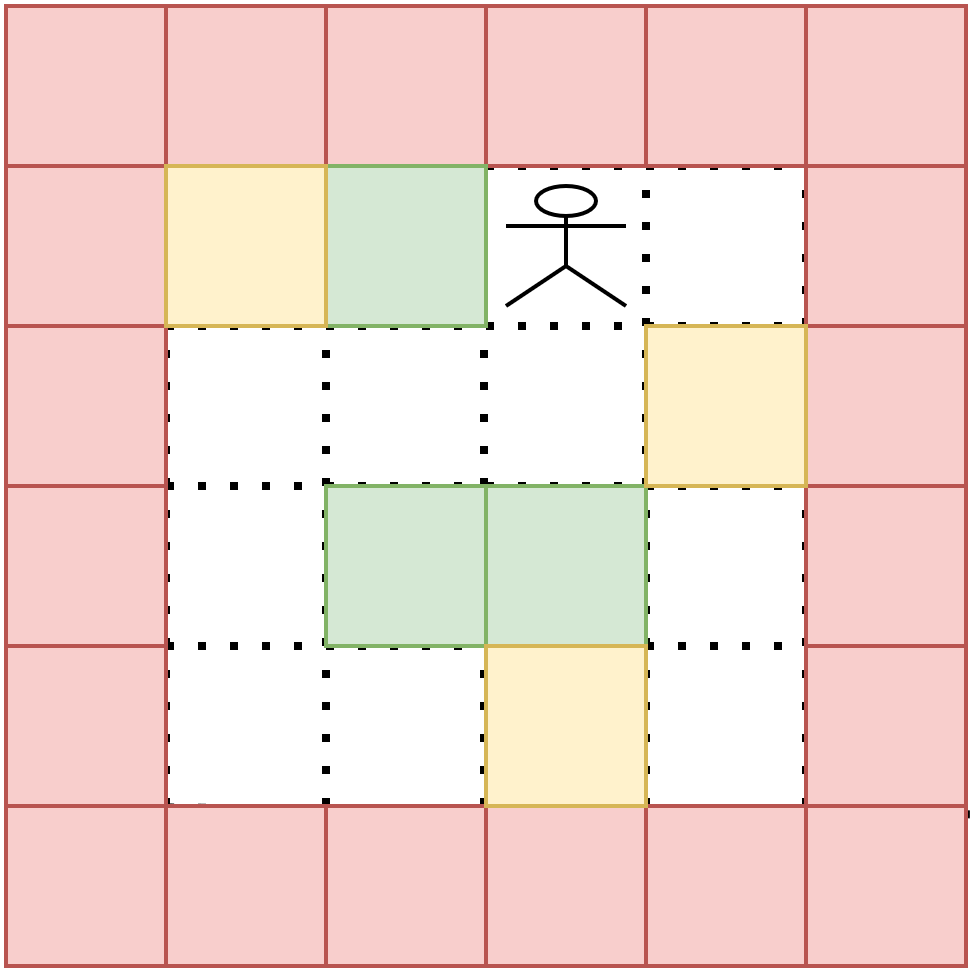
Sokoban is a transportation puzzle game created by Hiroyuki Imabayashi in 1981. Sokoban game solver logic has been applied in robotics for searching and motion planning in warehouses to move objects to their storage locations. Sokoban game is a NP-hard problem (Dor and Zwick 1996) and PSPACE-complete (Culberson 1997). The Sokoban game is played against the clock with the number of moves counted at every step. In gaming competitions, minimal moves with the least amount of time makes the player Win the game. Since the Crates and the Stores need to be located first before moving the Crates to the storage location, it can be alternatively looked at as a search problem and since the Crates need to moved to different location efficiently it also requires the path optimization.

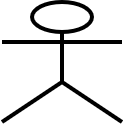
Basic game play or mechanism :

The game consists of 4 entities. The Player, Crates, Stores and a wall. All the entities are set up in a 2 dimensional grids. The grid consists of empty squares that can be a floor, a wall, a crate & a store. The start state is with crates, stores and the player occupying random locations on the grid. The goal state is when all the crates are moved to their respective stores. There are equal number of crates and stores. The solution is a sequence of actions (Move, Push) that transforms the game from Start state to a Goal state. The player has to push one crate at a time to its designated store working around the obstacles (wall or another crate) in its path until every crate has been pushed to its designated store. The pull operation doesn’t exist in the game setting. The player wins the game when it has pushed all the crates to its stores. The player loses the game or the game ends when it has reached a deadlock situation i.e. if the crate can’t be reached by the player or the crate is in a position that cannot be maneuvered around. So the game has a start state and an end state.

The motion of the player and crates can be either Up, Down, Left or Right (horizontal and vertical directions only). The player or the crate can move only one grid at a time provided the grid is an empty space. For example, let’s say, the crate is occupying position (x, y) and it needs to be pushed to the Right towards (x+1, y) position provided there isn’t an obstacle(wall or another crate). Then the player has to reach (x-1, y) position to execute the push operation if it is available and only then the crate can be pushed. After the crate has been pushed to the right by one grid, the crate will occupy (x+1, y) and the player will occupy (x, y) position on the grid after the push operation. Similar logic can be applied to pushing the crate in other directions. The player has two action states Move and Push. The Move is used to get adjacent to the crate that needs to be transported and Push is used to accomplish the task of transporting the crate to its final store position. The crates remain stationary until the player pushes them. The wall is an obstacle which the player or the crate can’t enter. Shown below is a simple layout of the Sokoban game setup.

Fig.1



 🡪PLAYER(**$**)  🡪WALL(**#**)  🡪CRATE(**B**)  🡪STORE(**.**)

A\* Implementation :

We have implemented the game by designating the character ‘$’ as the Player, ‘#’ as the wall and ‘B’ as the Crate, ‘ . ’ as the Store. When the crate has been transported to the store location the ‘ \* ’ character is used to indicate end of that action.

The factors that were considered during selection of an algorithm for this game were:

1. Optimal set of moves in a shorter amount of time (space & time complexity).
2. Constraints in the form of a wall/crate to the movement of crates and the player.
3. The depth of the solution space the algorithm can search for an optimal set of moves.

We implemented the Breadth First Search (BFS) and Depth First Search (DFS) to understand the space & time constraints of running the game. The number of moves made and the time taken to accomplish the task was observed.

Suggestions for further improvements :

References or Citations :

1. Dor, D., and Zwick, U. 1996. Sokoban and other motion planning problems. Computational Geometry 13(4):215–228.
2. Culberson, J. 1997. Sokoban is pspace-complete. In Proceedings in Informatics, volume 4, 65–76. Citeseer.
3. https://en.wikipedia.org/wiki/Sokoban