CAB320 – Assignment 1
Sokoban Assignment

John Santias Greyden Scott, Alex Holm

Sokoban is a puzzle game in which a player a warehouse worker, moving one or more crates from a starting position to a goal position. Created in 1981 by Hiroyuki Imabayashi, the game is played on a board of squares, where each square is a floor or a wall. In order for the player to relocate the crates to the goal, the player moves the warehouse worker around the game board pushing crates, providing there is floor space adjacent to the crate in the direction the warehouse worker is pushing. The player is restrained to only the following directions when pushing crates: left, right, up and down.

Notations

The notations used to represent the ware house in this report:

@	The player
#	A wall square
Space	A floor square
$	A crate
.	The goal square
!	The player on a goal square
*	A box on a goal square
X	A taboo floor square

Functions

1. taboo_cells

This function identifies taboo squares within the warehouse, returning a string that represents the warehouse, but only containing walls and taboo squares. A square is identified as taboo when it's identified as a square that players could push the crate on to, but then would not be able to make any addition moves due to the game play constraints.

This function finds taboo cells by first checking two rules. The first rule checks if the square is a corner, the second rule checks for all squares along a wall between two corners, unless the square is a goal. Once these two rules have been checked, the

function returns a string representing the puzzle containing only wall cells marked with a '#' and taboo cells marked with an 'X'.

Figure 1. Example of warehouse inputted into taboo_cells and the output returned

2. check_action_seq

The function check_action_seq is utilised to determine if a sequence of actions is possible. The function takes 2 parameters, a warehouse and a list of actions. The list must be constructed and passed in the following form: ['Right', 'Down', Left', 'Up',…].

For an action to be legal it must adhere to the specified criteria:

- The action must not push more than one crate at a time
- The action must not result in a crate being pushed into a wall
- The action must not move the walker into a wall

If an action is deemed legal, the function executes each action in the list and returns the resulting warehouse, otherwise it will return the string 'Failure'.

3. solve_sokoban_elem

This function solves the puzzle using elementary actions. An elementary action is defined as an action that the worker takes to push the crate to a target location. These actions are limited to: left, right, up and down.

This function takes a valid warehouse object as the parameter and returns a list containing the elementary actions to solve the puzzle.

This function operates by first generating a list of actions need to solve the puzzle by using the solve_sokoban_macro, as outlined in section 5. It then checks to see if a valid list of actions has been returned and then appends them to array and returns them. If the returned results from solve_sokoban_macro are invalid the function will return an empty array.

4. can_go_there

The purpose of this function is to determine whether the worker is able to move to a specified square in the warehouse. This function takes a valid warehouse object and returns a True of False Boolean.

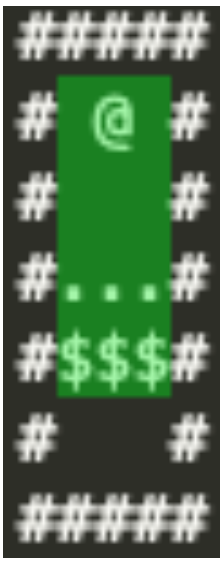

Figure 2. Green highlight is all the squares that would return true by can_go_there

5. solve_sokoban_macro

This function is used to determine the macro actions needed for solving the puzzle. A macro action xxx the location of a crate and the direction it has to be pushed in. This function takes a valid warehouse object as the parameter and returns a sequence of macro actions If the puzzle is solvable. Otherwise, if the puzzle is unable to be solved, it will return the string “Impossible”.

This function uses a helper function called SearchMarcroActions, which contains a number of additional functions to break down the result. In essence, this function searches through all valid squares the warehouse, checking if the worker can move to that location, for all those locations, the function then checks if the surrounding squares are valid moves, generating a list of the valid path’s the worker can take.
This list is than passed through an a* graph search to find the most efficient list of steps for the worker to move through the ware house.

Heuristic

As described in solve_sokoban_macro, we have utilised the A* graph search algorithm to find a list of steps for the worker to move through the warehouse with a crate to the goal. A* is an informed search algorithm, also known as a best-first search. A* starts from a specific starting node of a graph and aims to find a path to the given goal with the smallest cost. In our case, the smallest cost is calculated as the least distance travelled. This is done by maintaining a tree of paths that originates at the starting node and extending those paths one edge at a time until it reaches its goal.