Ho Fong Law(n10107321), Kiki Mutiara(n 10031014)

assignment 1

Sokoban puzzle

Problem Statement

In this assignment, we building a game called Push Box which is a computer puzzle game. Players can push boxes in the puzzle so that Place them in the designated position. It was originally revealed in 1982, which has since been realised in many computer platforms and video games comfort.

Not only it could be a game but design an algorithm for a robot that moves boxes in a warehouse. As the lecture said, the push box has been proven to have NP-hardness because of the depth of the solution. In other words, the puzzle does need to perform so much step to reach the goal state! Although it has limited branching factor with four directions (up, down, right, left). In simply, we must design a programme that Workers can only push one box at a time and cannot pull any boxes. If boxes reach the target location no matter the box identity, game finished, and the puzzle solved.

Structure

The programming structure using three main parts. The first part for checking Sokoban taboo cell which has two rules must follow. One is if the location is beside two walls. In the order word, a corner which will be labelled as a taboo cell. Another one is any locations and besides with a wall and no targets between those two corners. This would be also labelled as a taboo cell. taboo cell filters out the possible direction that the box could be pushed thus the searching depth would be limited to those possible movements and speed up the process.

The second part is the main Sokoban class, in here, it performs most of the actions in it with provided functions. The walkthrough would be first getting a goal test to see whether the puzzle is already finished during the start. Then start to develop a possible action if the move is valid. Later, return a result for substitute into the child node which expands by a parent node.

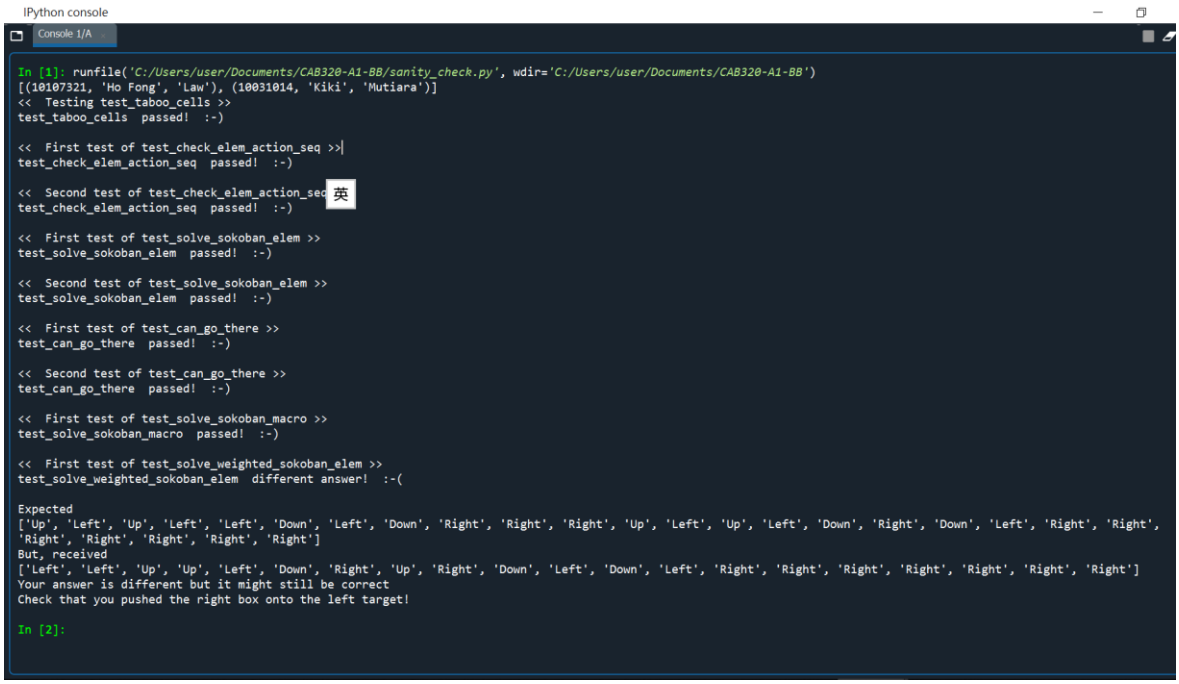
Third parts are searching for selection. We have three functions for performing different ways for search. Firstly, element search performs the basic search method which handles worker movement and each movement cost would consider as 1. This searching method have no priority of which box be pushed first. Instant, function focus on Manhattan Distance calculated in h value. Secondly, Macro focus on the cost when worker actual pushes the boxes. Lastly, Weight element functions would stack on the weighted of boxes and see which boxes with the lowest cost and Manhattan distance.

Implementation

Function Name	Parameter	Input	Output	Algorithm
my team			Team name and number	
taboo_cells	Warehouse	Warehouse class type	A string that represent the puzzle taboo locations with wall	
taboo_reader	Taboo_cell_str	A string that represent the puzzle taboo locations with wall	A list that represent taboo position	
SokobanPuzzle: __init__, Actions, Result, Goal_test, Path_cost, H,	Search.Problem; (self, initial=None, allow_taboo_push=None, macro=None,push_costs=None); (self, state); (self, state, action); (self, state);	The Sokoban puzzle in warehouse type; Initial warehouse type puzzle, allow taboo push or not, macro push or not, what the push cost;	Class that represent, Sokoban; Initial data we need. Return action we could perform.	

	(self, c, state1, action, state2); (self, n);	String type warehouse. String type warehouse, action need to perform; String type warehouse. C is original cost, String type warehouse, action would perform, String type warehouse after action; Node.	Return result that state after action; Check whether reach to goal; Return the path cost after performing action; Return how far Manhattan distance between two closest points	
check_elem_action_seq	warehouse, action_seq	Warehouse type puzzle, action sequence that need to perform	Return actions sequence are valid or not	
solve_sokoban_elem	warehouse	Warehouse type puzzle	Return actions sequence	Element movement
can_go_there	warehouse, dst	Warehouse type puzzle, distance location	Return destination is valid to go there or not	
solve_sokoban_macro	warehouse	Warehouse type puzzle	Return location of boxes and actions sequence	Macro movement
solve_weighted_sokoban_elem	warehouse,push_costs	Warehouse type puzzle, push costs of each box	Return actions sequence	Weighted element movement
Pointer: __init__, heap, __str__, move_to	[]; (self, name, heap); (self); (self); (self, position);	[]; Name of way and number have to heap. []; []; Position need to be change.	[]; []; Return heap. Return name. Return position after heap it on;	
TempSokuban: __init__, Actions, Result, H	search.Problem (self, initial, goal, warehouse) ; (self, state); (self, state, step); (self, n);	The Sokoban puzzle in warehouse type. Initial warehouse type puzzle, goal warehouse type puzzle; warehouse type puzzle. warehouse type puzzle, step that need to perform; node.	[]; []; Return list of actions. Return position. Return	
manhattan_distance	loca_a, loca_b	Location A and location B that need to compare	Return Manhattan distance between two points	

Testing Section



Epilogue

About the weigh elementary move, output result is not as excepted. The possible reason should because the h and weight value are not ideal for adjust what actual value should be. That could lead to non-optimistic solve result.

Expected

['Up', 'Left', 'Up', 'Left', 'Left', 'Down', 'Left', 'Down', 'Right', 'Right', 'Right', 'Up', 'Left', 'Up', 'Left', 'Down', 'Right', 'Down', 'Left', 'Right', 'Right', 'Right', 'Right', 'Right', 'Right', 'Right']

But, received

['Left', 'Left', 'Up', 'Up', 'Left', 'Down', 'Right', 'Up', 'Right', 'Down', 'Left', 'Down', 'Left', 'Right', 'Right', 'Right', 'Right', 'Right', 'Right', 'Right']