DSECLZG557 – Assignment #1

ACI Assignment Group: 41

1. Formulate the Bloxorz problem as a search problem by depicting its states representation, initial state, actions, transition model, goal state.

States Representation (each state of the problem): Position of the block in the terrain (x1, y1, x2, y2). Since the block is of two bricks it is represented by two (x, y) coordinates, where x1, y1 is the starting tile and x2, y2 is the ending tile where the block lies in a given state. The block can be in three states – standing, lying horizontally and lying vertically.

Initial/ Start State (the state in which the agent exists initially): (2, 2, 2, 2) or as a user determines it. It is denoted by **S**.

Actions (possible actions used to solve the problem): Block can move in four directions – right, left, up and down to go to a new state. According to the game rules it is to be ensured that only legal moves are allowed - the ends of the block are always within the terrain boundary, until it reaches the goal state.

Transition Model (description of what each action does): Return the new legal position of the block after applying any legal move, as per the game rules.

Goal State (desired result in a problem): Position (5, 8, 5, 8) of the block in vertical state, such that it falls inside the hole. It is denoted by **G**.

State Space (of the game problem): The set of all possible states that are possible in the problem. It comprises of the initial state along with the actions and the transition model.

Path Cost function g(n): The path cost of any state/ node n, is the sum of costs of individual actions/ steps along the path from the initial state to that state/ node n. Here step cost is 1 for every move.

1. Can BFS/ DFS be used to solve this problem? If so, explain how it can be used by providing an algorithm/ pseudocode.

Yes, Bloxorz puzzle game can be solved thru a single agent using BFS/ DFS (uninformed search) algorithm by categorizing as a path-finding problem. We would need two data structures, ‘Expand’ (to store the nodes to be expanded) and ‘Visited’ (to store the positions/ nodes the block has already been in) to keep track of already expanded/ visited nodes.

BFS is a strategy for exploration that always selects the shallowest fringe node from the start node for expansion, for which the ‘Expand’ data structure is a FIFO (first-in-first-out) Queue. DFS is a strategy for exploration that always selects the deepest fringe node from the start node for expansion, for which the ‘Expand’ data structure is a LIFO (last-in-first-out) Stack.

Pseudocode of algorithm for Breadth-First Search strategy:

1. Predefine the initial node (initial position of the block) and the goal node (position of the hole).
2. Initialize the MoveSequence string of the initial node to empty.
3. Add the initial node into the Expand queue and the Visited list.
4. Check if the queue is empty → there is no solution and exit, otherwise continue.
5. Poll out the first node of the Expand queue (called parent node).
6. Check if it is equal to the goal node (block position defined in this node equals hole position defined in the goal node) → solution is found and return the MoveSequence string of that node, otherwise continue.
7. Expand the node and generate its child nodes which represent the new positions of the block after moving in all 4 directions from the current position defined in the parent node.
8. For each of the generated nodes, if it is valid (block position is valid in the terrain), its MoveSequence equals to its parent’s plus the move it generated from, add the node to the Expand queue and the Visited list.

Finally go back to step iv.

Pseudocode of algorithm for Depth-First Search strategy:

1. Predefine the initial node (initial position of the block) and the goal node (position of the hole).
2. Initialize the MoveSequence string of the initial node to empty.
3. Add the initial node into the Expand stack and the Visited list.
4. Check if the stack is empty → there is no solution and exit, otherwise continue.
5. Pop out the top node of the Expand stack (called parent node).
6. Check if it is equal to the goal node (block position defined in this node equals hole position defined in the goal node) → solution is found and return the MoveSequence string of that node, otherwise continue.
7. Expand the node and generate its child nodes which represent the new positions of the block after moving in all 4 directions from the current position defined in the parent node.
8. For each of the generated nodes, if it is valid (block position is valid in the terrain), its MoveSequence equals to its parent’s plus the move it generated from, add the node to the Expand stack and the Visited list.

Finally go back to step iv.

1. Can A\* search be used to solve this problem? If so, explain how it can be used by providing an algorithm/ pseudocode.

Yes, Bloxorz puzzle game can be solved thru a single agent using A\* (informed search) algorithm by categorizing as a path-finding problem. We would need a data structure ‘Expand’ (to store the nodes to be expanded). The heuristic value/ evaluation function would use Manhattan distance as the heuristic, to guide the search process.

A\* Search is a strategy for exploration that always selects the fringe node with the lowest estimated total cost for expansion, for which the ‘Expand’ data structure is a Priority Queue. Its priority is based on an evaluation function f(n) and the highest priority is for a node with the minimum f(n) value.

**f(n) = g(n) + h(n)**, where:

g(n) – The function representing total backward cost, which is actual path cost from the Initial/ Start state/ node S to the state defined in the current state/ node n.

h(n) – The heuristic value/ evaluation function representing estimated forward cost, which is estimate path cost from the state defined in current node n to reach the Goal state/ node G.

f(n) – The function representing estimated total cost, which is overall path cost from the initial state/ node S passing by the state in node n to the goal state/ node G.

Since the block has two bricks and is represented by two (x, y) coordinates, the heuristic value for both bricks h(n1) and h(n2) are calculated, and the maximum one is considered as the overall heuristic.

**h(n) = max[h(n1), h(n2)]**, where n1, n2 are the two bricks of the block.

**h(ni) = max(|xi-xg|, |yi-yg|)**, where i = {1, 2} for the two bricks of the block;

(xi, yi) is the current position of brick i;

(xg, yg) is position of goal state.

Pseudocode of algorithm for A\* Search strategy:

1. Predefine the initial node (initial position of the block) and the goal node (position of the hole).
2. Initialize the MoveSequence string of the initial node to empty.
3. Add the initial node into the Expand queue.
4. Check if the queue is empty → there is no solution and exit, otherwise continue.
5. Poll out the highest priority node of the Expand queue (called parent node).
6. Check if it is equal to the goal node (block position defined in this node equals hole position defined in the goal node)→solution is found and return the MoveSequence string of that node, otherwise continue.
7. Expand the node and generate its child nodes which represent the new positions of the block after moving in all 4 directions from the current position defined in the parent node.
8. For each of the generated nodes, if it is valid (block position is valid in the terrain), its MoveSequence equals to its parent’s plus the move it generated from, compute its *h(n)* value and *g(n)* which is equal to its parent’s plus 1, then set its *f(n) = h(n) + g(n)*, add the node to the Expand queue. Finally go back to step iv.
9. Implement an agent to solve level-1 of Bloxworz game.