Term Project: (This term project will be divided into several modules (i.e., assignments). We will use what we have learnt from our lecture notes to design and implement an intelligent agent. Using any existing implemented maze-problem-solver solution does not serve the course learning outcomes, and thus is prohibited.

Project description:

Consider a problem of developing (designing and implementing) an intelligent agent called the maze-problem-solver (or called it a mice).

Given ANY maze configuration, the maze has an entrance (equivalently, called the initial state) for entering the maze, and two exits, g1 and g2 (or called final states). The maze-problem-solver can find its ways from entrance **[ ],** to an exit **] [.** The goal is that the maze-problem-solver enters the given maze and find a way for exiting from the maze with minimal time span (in terms of number of **grids** traversed and **turns**).

A maze configuration is given at the end of this assignment. We will continue to use this maze configuration for the remaining assignments (i.e., project).

Background

Review the solution for the problem Q1 of the assignment As01; and the solution described all the possible distinct states of the percepts for the given maze configuration and the possible admissible (or primitive) actions in the form of action(state, state). The primitive, admissible actions include only forward(state, state), rightTurn(state, state) and leftTurn(state, state). (The recommended solution has been posted on the brightspace.). An action is a mapping from a state to another state. It is possible for the resulting state of an action be at the same grid. For example, both rightTurn and leftTurn yield another state, where the maze-problem-solver is still remaining at the same grid, but the solver heads into different direction. Per our discussion in class, a state can be described in terms of (top, right, back, left)-tuple. Implicitly, top means the front(top) of a maze-problem-solver (or called a mice), its right, its tail(back) and its left. Therefore, the same grid can have at most 4 ways of specification for a state. Therefore, a state of percepts for a grid could be specified at most 4 ways. (See the provided document CS 380 StateSpace 02092021 in Brightspace.) In any given maze, the configuration of grids can be classified into a finite number of types based on assigning a value 2 for open(no wall, or grid line), 1 for wall, 0 for entrance/exit for each grid in the given maze. For example, if the mice (the maze-problem-solver) enters into grid A of the given maze, the configuration of the grid A is (2, 2, 0, 1); this means top(mice’s front) is 2, (mice’s) right is 2, (mice’s) back is 0, and (mice’s) left is 1. At grid A, the mice can take a sequence of actions, such as rightTurn((2, 2, 0, 1), (2, 0, 1, 2)), then rightTurn((2, 0, 1, 2), (0, 1, 2, 2)), then another rightTurn((0, 1, 2, 2), (1, 2, 2, 0)). This depicts that the grid A can have 4 ways of specifications (2, 2, 0, 1), (2, 0, 1, 2), (0, 1, 2, 2), (1, 2, 2, 0), depending upon where the mice heading to (where the sensors of the mice at). Any one of these four specifications for the grid A, the configuration of the A depicts uniquely as follows:

 grid A’s specification could be (2, 2, 0, 1)

Thus, the state of percepts for the grid A is one of these 4 ways of specifications for the grid A. The state of a percept for the grid A is (2, 2, 0, 1) to refer the same grid. We also can use the other three grid’s specifications to refer the state of percepts for the grid A.

At grid A, the mice could move forward through an avenue consisting a number of grids (the configuration of each grid is (2, 1, 2, 1)) before arriving at grid B. The state of percepts for these grids could be (2, 1, 2, 1). These grids have identical configuration with four ways of specification (2, 1, 2, 1), (1, 2, 1, 2), (2, 1, 2, 1) and (1, 2, 1, 2) depending on where is the front of the mice. Furthermore these grids has repeated (2, 1, 2, 1) and (1, 2, 1, 2) specification. Without loss of generality, the state of percepts for any of these grids (in the avenue) can be (2, 1, 2, 1) or (1, 2, 1, 2). Pictorially,

 The specification of this grid could be (2, 1, 2, 1).

Thus, the state of a percept for a grid in this avenue is (2, 1, 2, 1) to refer the any grid of this avenue.

Once it arrives at the grid B, the specification of the grid B is (1, 2, 2, 1). Applying an action rightTurn at the grid B, turnRight((1, 2, 2, 1), (2, 2, 1, 1)), and turnRight((2, 2, 1, 1), (2, 1, 1, 2)), and again turnRight((1, 1, 2, 2), (1, 2, 2, 1)). The mice is still at the same grid B but different direction. This implies that the specifications for the grid B are (1, 2, 2, 1), (2, 2, 1, 1), (2, 1, 1, 2) and (1, 1, 2, 2). Likewise, it would be the same for an action turnLeft. Pictorially,

 The specification of the grid B could be (1, 2, 2, 1).

The state of percepts for grid B is one of these 4 ways of specifications for the grid B. The state of a percept for the grid B is (1, 2, 2, 1) to refer the same grid B.

In conclusion, the question is how do you specify states for a given maze. Given ANY maze, it has a limited number of grid types, depending upon that the side of a grid is a wall (specifies as 1), open (specifies as 2), and an entrance/exit (specifies as 0). Each of the grid types can have four ways of specifications. Per grid type, choose arbitrarily one of the specifications for the grid type to be the state of percepts for the grid. Therefore, the number of possible distinct state of percepts for grids is a limited (7 distinct states?). Without loss of generality, we could define the state of the percept for the mice outside the entrance is (0, -, -, -), and the state of the percept for the mice outside the exit is (-, -, 0, -).

The previously given assignment As02 consists of three questions I through III. In this assignment, the given problem is formulated as a search problem using a labelled graph based on a given maze configuration. The graph contains vertices and (directed) edges. A vertex represents one or more grids and has an inner graph which represents the mice’s position at a grid (See the document CS 380 StateSpace 02092021). Based on the result obtained in this assignment As01, any vertex is labelled by (top, right, back, left)-tuples for the grids. (It should not be labelled by any alphabets A – Z, or numeric 1-45, as given in the maze configuration.) Each edge represents a transition between states. Associated with an edge, there is a defined action, either forward, rightTurn or leftTurn. We will examine a number of algorithms such as breadth-first search, uniform cost search and many others used to traverse this graph finding its exit (either grid g1 or grid g2) after entering at grid A. The traversal of the graph from initial state forms paths from a root node through nodes; thus, a tree-search is formed. From programming point of view, this graph can be implemented; otherwise, there need hand-executing algorithms applied on this graph.

The basic question is that how do you formulate the problem as a labelled graph. Each vertex represents a possible state; a vertex is designated as the initial state and two vertices are designated as goal states; each edge represents a state transition triggered by a specific agent action. Associated to each edge is the cost of performing that transition. The labelled graph implicitly has a search space, which is a set of states reachable from an initial state (2, 2, 0, 1) via a (possibly empty, finite or infinite) sequence of state transitions. Based on the search space of the given labelled graph, allows to *search* the space for a possibly optimal sequence of state transitions starting from the initial state (2, 2, 0, 1) to a goal state (0, 1, 2, 1) by *orderly* *executing* the actions associated to each transition in the identified sequence of state transitions. Thus, the given problem for formulating a labelled graph in terms of search space is reduced to a search problem of the search space. Then the solution for the search problem is *finding* a path from an initial state to a goal state according to orderly execution of the sequence of actions associated with the *identical* path and the description of the goal state. Successful construction of the search space becomes critical issue for applying various algorithms for finding solutions. For example, how do you construct a state n in the state space STATE to which the node corresponds, and thus

corresponds a grid configuration for the given maze. Your assignment As02 has addressed this issue. For example, how do you specify that the states of percepts for grids n, A and B in given maze configuration corresponds to the nodes (vertices) in the state space STATE.

 the state space STATE

grid A

n.PARENT: The parent node generates

a child node associated with state n.

 state

 n.ACTION: At node A, take

 forward to reach a child node

 associated with state n.

 a node

grid n

 state

 n.PATH-COST: The cost g(n) of the

 path from the initial state to the node n.

grid B

 state

Based on the assignment As01, these three grids has their specifications, and therefore, the states of percepts for these three grids n, A and B could be (2, 1, 2, 1), (2, 2, 0, 1) and (1, 2, 2, 1) respectively. This approach could be used to describe the associations of the state of a percept for grid n and a node, the state of a percept for grid A and a node, and the state of a percept for grid B and a node in a search state space, STATE as follows:

 forward((2, 2, 0, 1), (2, 1, 2, 1))

forward((2, 1, 2, 1), (2, 1, 2, 1))

forward((2, 1, 2, 1), (1, 2, 2, 1))

The association between a node in the search state space, STATE and the state of percept for a given grid could be described clearly and precisely accordingly. However, this is inadequate for there is 4 ways of specification at each grip. For the grid n in an avenue between grid A and grid B, there is an inner graph, which represents that the maze-problem-solver takes actions leftTurn and rightTurn at grid n: This inner graph for the node n would be as follows:

 rightTurn

 leftTurn

 rightTurn rightTurn

 leftTurn leftTurn

Equivalently this inner graph for this particular grid configuration can simplify as

 rightTurn

 leftTurn

This state of percepts for grid n (in the avenue) in the state space STATE therefore can be represented as:

rightTurn

leftTurn

Likewise, the state of percepts for grid A in the state space STATE therefore can be represented as:

 rightTurn

[2, 2, 0, 1] [2, 0, 1, 2]

 leftTurn

rightTurn rightTurn

 leftTurn leftTurn

 leftTurn

[1, 2, 2, 0] [0, 1, 2, 2]

 rightTurn

The actions can be specified as: note that the node (vertex) of the state space STATE is without label, but the state of the percepts for its associated grid with actions.

 forward((2, 2, 0, 1), (2, 1, 2, 1))

forward((2, 2, 0, 1), (2, 1, 2, 1))

 rightTurn

[2, 1, 2, 1] [1, 2, 1, 2]

 leftTurn

 forward((2, 1, 2, 1), (1, 2, 1, 1))

 rightTurn

[1, 2, 2, 1] [2, 2, 1, 1]

rightTurn rightTurn

 leftTurn leftTurn

 leftTurn

[1, 1, 2, 2] [2, 1, 1, 2]

 rightTurn

 forward((2, 1, 1, 2), (2, 1, 2, 1))

forward((2, 1, 2, 1), (2, 1, 2, 1))

So far, the maze-problem-solver problem is formulated into labelled graph, then to the search state space. Using the introduced specification technique for the grids of any given maze configuration, we unify the states of percepts for the grids, the associations between grids in the given maze configuration and the nodes (vertices) of the search state space from a labelled graph, and that corresponding to each transition, there is an action associated with the transition as a whole - the search state space The STATE, without using artifact label such as A - Z and 1 - 45 .

The assignment As03 is continuation of assignments As01 and As02. Based on the obtained results from the previous two assignments As01 and As02, assignment As03 is given as follows:

In the above description, based on the search space of the formulated, labelled graph, allows to *search* the space for a possibly optimal sequence of state transitions starting from the initial state (2, 2, 0, 1) to a goal state (0, 1, 2, 1) by *orderly* *executing* the actions associated to each transition in the identified sequence of state transitions. Thus, the given problem for formulating a labelled graph in terms of search space is reduced to a search problem of the search space. Then the solution for the search problem is *finding* a path from an initial state to a goal state according to orderly execution of the sequence of actions associated with the *identical* path and the description of the goal state. In other words, let develop algorithms for traverse the search state space STATE to from tree search for the solution (i.e., a path from the root which corresponds to an initial state) to a node which is a goal state. (In fact, the STATE is a graph labelled by the state of percepts for grids in the maze)

In assignment As03, based on this obtained search state space (the STATE – it is a graph), design and implement a breadth-first search and uniform cost search algorithms on the STATE obtained in assignment As02, based on the given algorithms in given lecture note Ch 03\_02\_fs Uniformed Search as below. That is, you are required to implement these two algorithms, where the input is the search state space you have obtained from your assignments As01 and As02.

function BREADTH-FIRST-SEARCH(problem) returns a solution, or failure

node $\leftarrow $ a node with STATE = problem.INITIAL-STATE, PATH-COST = 0;

if problem.GOAL-TEST(node.TEST) then return SOLUTION(node);

frontier $\leftarrow $ a FIFO queue with node as the only element;

explored $\leftarrow $ an empty set; //use hash table?

loop do

 if EMPTY?(frontier) then return failure;

 node $\leftarrow  $POP(frontier); //chooses the shallowest node in frontier

 add node.STATE to explored;

 for each action in problem.ACTION(node.STATE) do

 child $\leftarrow  $CHILD-NODE(problem, node, action);

 if child.STATE is not in explored or frontier

 then if problem.GOAL-TEST(child.STATE)

 then return SOLUTION(child);

 frontier $\leftarrow   $INSERT(child, frontier);

Figure 3.11 Breadth-first search on a graph

function UNIFORM-COST-SEARCH(problem) returns a solution, or failure

node $\leftarrow $ a node with STATE = problem.INITIAL-STATE, PATH-COST = 0

frontier $\leftarrow $ a priority queue ordered by PATH-COST, with node as the only element

explored $\leftarrow $ an empty set

loop do

 if EMPTY?(frontier) then return failure

 node $\leftarrow  $POP(frontier) //chooses the lowest-cost node in frontier

 if problem.GOAL-TEST(node.TEST) then return SOLUTION(node)

 add node.STATE to explored //signified the node is visited

 for each action in problem.ACTION(node.STATE) do

 child $\leftarrow   $CHILD-NODE(problem, node, action)

 if child.STATE is not in explored or frontier then

 frontier $\leftarrow   $INSERT(child, frontier)

 else if child.STATE is in frontier with higher PATH-COST then

 replace that frontier node with child

Figure 3.14 Uniform-cost search on a graph

**Results**

**This assignment As03 must have the following write-ups:**

**Q1. Describe briefly the data structure of the fringe (or frontier, open list) if you used for each of the algorithms. Provide an evidence (histories) how it is used for each algorithm on the search state space.**

**Q2. Describe briefly the data structure of the exploration (or explored set, closed list) if you used for each of the algorithms. Provide an evidence (histories) how it is used for each algorithm on the search state space.**

**Q3. Describe briefly how do you calculate the path-cost for each of the two algorithms. What are the assumptions, if there are, do you use?**

**Q4. Provide SOLUTION(node), which are paths from the root to node. These paths are obtained from searching for the transition sequence from initial state to goal states and executing sequences of actions associated with transitions, using these two algorithms.**

**Q5. Evaluate the search strategy for each of the two algorithms in terms of solution completeness, time complexity, space complexity and optimality.**

**Q6. In tree search on the obtained graph (the state space – the STATE), each node of a tree has its associated state (which is the state of percepts for a grid). Is there a simple way for constructing this association between a node in a state space and its associated state of the given grid configuration in the given maze, especially which is needed in the programming?**

**The following graph is a maze example: Assume A is the entrance. g1 and g2 are exits.**

**] [**

**] [**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 41 |  |  |  |  |  | 42 |  |  |  |  |  |  | 43 |  |  |  |  | 44 | **g1** |
| 40 |  |  |  |  |  | 39 |  |  | 38 |  |  |  | 37 |  |  |  |  | 36 |  |
|  |  |  |  |  | 35 |  |  |  |  | 34 |  |  | 33 |  |  |  | 32 |  | 31 |
| **D** |  |  |  | **M** | **Q** |  |  |  |  |  |  | 9 |  |  |  | 20 | 23 | 27 | 30 |
|  |  |  | **I** |  |  |  |  |  | 2 |  |  |  | 13 |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **C** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  | **1** |  | 5 | 8 | 12 |  | 15 |  |  |  |  |
|  |  | **E** | **H** | **L** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **B** |  |  |  |  |  | **R** |  | **X** |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  | **P** |  | **T** |  |  |  |  |  |  |  |  |  | 22 | 26 | 29 |
|  |  |  | **G** | **K** | **O** |  |  |  | **Z** | 3 | 4 |  |  |  |  | 19 | 21 | 25 | 28 |
|  |  |  |  |  |  |  |  | **W** |  |  |  | 7 | 11 |  |  |  |  |  |  |
|  |  |  |  |  |  |  | **S** |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  | 14 |  | 18 |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  | **V** |  |  |  | 6 |  |  |  | 17 |  |  |  |
|  |  |  |  |  |  |  |  | **U**  |  |  |  |  | 10 |  |  |  |  |  |  |
| **A** |  |  | **F** | **J** | **N** |  |  |  | **Y** |  |  |  |  |  | 15 | 16 |  | 24 | g2 |

**[ ]**

