|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Pratik Pujari** | | |
| **UID no.** | **2020300054** | **Class:** | **Comps C Batch** |
| **Experiment No.** | 1 | | |

|  |  |
| --- | --- |
| **AIM:** | Implement an Intelligent agent.  (problem formulation and implementation) |
| **WATER JUG PROBLEM** | |
| **FPRMULATION:** | Problem statement  Given two water jugs with capacities **X** and **Y** litres. Initially, both the jugs are empty. Also given that there is an infinite amount of water available. The jugs do not have markings to measure smaller quantities. One can perform the following operations on the jug:   * Fill any of the jugs completely with water. * Pour water from one jug to the other until one of the jugs is either empty or full, (X, Y) -> (X – d, Y + d) * Empty any of the jugs   The task is to determine whether it is possible to measure **Z** litres of water using both the jugs.  **Agent Function:**   |  |  |  |  | | --- | --- | --- | --- | | **Sr.No** | **Description of action taken** | **Condition** | **Final state** | | 1. | Fill the j1-litre jug completely | If x<j1 | (j1,j2) | | 2. | Fill the j2-litre jug completely | if y<j2 | (x,j2) | | 3. | Empty the j1-litre jug | If x>0 | (0,y) | | 4. | Empty the j2-litre jug | If y>0 | (x,0) | | 5. | Pour some water from the j2-litre jug to fill the j1-litre jug | If (x+y)<j1+j2 | (j1, y-[j1-x]) | | 6. | Pour some water from the j1-litre jug to fill the j2-litre jug | If (x+y)<j1+j2 | (x-[j2-y],j2) | | 7. | Pour all water from j2-litre jug to the j1-litre jug | If (x+y)<j1 | (x+y,0) | | 8. | Pour all water from the j1-litre jug to the j2-litre jug | if (x+y)<j2 | (0, x+y) |   Agent: Water-Jug Agent  Action: fill jug, empty jug, transfer from one jug to other  Goal : x1 litres water in X jug and y1 litres water in Y jug  Environment Type: single agent, partially observable, deterministic, static, Discrete  PEAS Description:  1.Performance measure:  Correctness i.e. successfully reach goal state  2.Environment:  Two jugs and a tap with unlimited water  3.Actuators:  The Pump, to transfer the water  4.Sensors:  Sensor to detect Water level  The agent has sensors to judge when when jug is empty and full.  Ex: If max X = 4, max B = 3 and final x=2 final y=0   |  |  |  | | --- | --- | --- | | **4 litres jug** | **3 litres jug** | **Rule Applied(to get 2 litres)** | | 0 | 0 | Initial State | | 4 | 0 | 1.Fill 4 | | 1 | 3 | 6. Pour 4 into 3 to fill | | 1 | 0 | 4. Empty 3 | | 0 | 1 | 8. Pour all of 4 into 3 | | 4 | 1 | 1. Fill 4 | | 2 | 3 | 6.Pour into 3 | | 2 | 0 | Empty 3 |   State Space Graph |
| **IMPLEMENTATION:** | *Code:*  import java.util.ArrayList;  import java.util.LinkedList;  import java.util.Queue;  import java.util.Scanner;  class Node {      int leftCap, rightCap;      String path;      public Node(int x, int y, String z) {          this.leftCap = x;          this.rightCap = y;          this.path = z;      }  }  public class WaterJugProblem {      public static int jugA, jugB, targetA, targetB;      public static Queue<Node> queue = new LinkedList<>();      public static ArrayList<Node> visited = new ArrayList<>();      public static int visitedNodes = 0;      public static boolean flag = false;      public static void main(String[] args) {          Scanner sc = new Scanner(System.in);          System.out.print("Enter the Capacity of Jug A: ");          jugA = sc.nextInt();          System.out.print("Enter the Capacity of Jug B: ");          jugB = sc.nextInt();          System.out.print("Enter the Target Capacity of Jug A: ");          targetA = sc.nextInt();          System.out.print("Enter the Target Capacity of Jug B: ");          targetB = sc.nextInt();          Node root = new Node(0, 0, "");          queue.add(root);          Node jug;          while (!queue.isEmpty()) {              visitedNodes++;              jug = queue.poll();              if (jug.leftCap == targetA && jug.rightCap == targetB) {                  System.out.println("The required capacity can be achieved after " + (visitedNodes + 1) + " move(s)");                  System.out.println("Number of nodes traversed: " + visitedNodes);                  System.out.println("The path is: " + generatePath(jug));                  flag = true;                  break;              }              Node temp = jug;  *// Fill jug A*              if (jug.leftCap < targetA) {                  temp = new Node(jugA, jug.rightCap, generatePath(jug));                  if (!visited.contains(temp)) {                      queue.add(temp);                      visited.add(temp);                  }              }  *// Fill Jug B*              if (jug.rightCap < targetB) {                  temp = new Node(jug.leftCap, jugB, generatePath(jug));                  if (!visited.contains(temp)) {                      queue.add(temp);                      visited.add(temp);                  }              }  *// Empty Jug A*              if (jug.leftCap > 0) {                  temp = new Node(0, jug.rightCap, generatePath(jug));                  queue.add(temp);                  visited.add(temp);              }  *// Empty Jug B*              if (jug.rightCap > 0) {                  temp = new Node(jug.leftCap, 0, generatePath(jug));                  queue.add(temp);                  visited.add(temp);              }  *// Pour from Jug A to Jug B until its full*              if (jug.leftCap > 0 && (jug.leftCap + jug.rightCap) >= jugB) {                  temp = new Node(jug.leftCap - (jugB - jug.rightCap), jugB, generatePath(jug));                  if (!visited.contains(temp)) {                      queue.add(temp);                      visited.add(temp);                  }              }  *// Pour from Jug B to Jug A until its full*              if (jug.rightCap > 0 && (jug.leftCap + jug.rightCap) >= jugA) {                  temp = new Node(jugA, jug.rightCap - (jugA - jug.leftCap), generatePath(jug));                  if (!visited.contains(temp)) {                      queue.add(temp);                      visited.add(temp);                  }              }  *// Puor all water from 1st to 2nd*              if (jug.leftCap > 0 && (jug.leftCap + jug.rightCap) <= jugB) {                  temp = new Node(0, jug.leftCap + jug.rightCap, generatePath(jug));                  if (!visited.contains(temp)) {                      queue.add(temp);                      visited.add(temp);                  }              }  *// Puor all water from 2nd to 1st*              if (jug.rightCap > 0 && (jug.leftCap + jug.rightCap) <= jugA) {                  temp = new Node(jug.leftCap + jug.rightCap, 0, generatePath(jug));                  if (!visited.contains(temp)) {                      queue.add(temp);                      visited.add(temp);                  }              }          }          if (!flag)              System.out.print("\nCannot achieve the required capacity");          sc.close();      }      static String generatePath(Node node) {          String path = "[ " + node.leftCap + "," + node.rightCap + " ] ";          return node.path + path;      }      static void printVisitedNodes() {          System.out.println("Visited Nodes: ");          for (Node node : visited) {              System.out.print(" [ " + node.leftCap + " " + node.rightCap + " ], ");          }      }  } |
| **CONCLUSION: Solved Water jug Problem using the BFS searching technique.** | |

|  |  |
| --- | --- |
| **MISSIONARY CANNIBAL PROBLEM** | |
| **FORMULATION:** | Problem Statement  In the missionaries and cannibals problem, three missionaries and three cannibals must cross a river using a boat which can carry at most two people, under the constraint that, for both banks, if there are missionaries present on the bank, they cannot be outnumbered by cannibals (if they were, the cannibals would eat the missionaries). The boat cannot cross the river by itself with no people on board.  PEAS   * Performance Measure:1.+100 for every correct placed cannibal or missionary.2.-50 when cannibal eats missionary. * Environment:3 cannibals & 3 missionary Boat, Left and Right banks of the river Cannibals and missionary are to be taken from the left corner of the bank to the right corner of the river without letting the cannibal eat the missionary. * Actuators: A boat to sail from one side of the river to the other * Sensors: To check whether the cannibals are more than the missionary |
| **IMPLEMENTATION:** |  |
| **CONCLUSION:** | |