|  |
| --- |
| Описание: https://lh6.googleusercontent.com/QcftzNtI05T0Y6fjdSh1Rr2rt8oqZ1IvnLvbn1jLJ7CCyteVir3k-xBLv4SL1wAgWJsRhmmJSR0UW-RP63_GQenE4vVWv05BRoZTsmIcBccVTnfxwmsnNMvjg599x9SqZd8E3dkd |
| МИНОБРНАУКИ РОССИИ |
| Федеральное государственное бюджетное образовательное учреждениевысшего образования"МИРЭА - Российский технологический университет"РТУ МИРЭА |

Институт информационных технологий

Кафедра вычислительной техники

**ОТЧЕТ ПО ЗАЧЕТНОЙ РАБОТЕ**

**по дисциплине**

**«АПО»**

Студент группы ИВБО-07-15 Боязитов И. Э.

Принял Смирнов С. С.

Москва 2019

Содержание

[1. Постановка задачи. 3](#_Toc6522128)

[2. Листинг программы. 4](#_Toc6522129)

[3. Результат работы программы. 15](#_Toc6522130)

1. Постановка задачи.

Разработать модель работы гипотетического метрополитена, в котором существуют среда и следующие типы агентов:

- метро;

- поезд;

- станция;

- пассажир.

Сначала среда (файл MetroEnv.java) создает граф с заданным количеством линий метро, заданным количеством станций на каждой линии, а также с пересадками, являющимися пересечением некоторых станций (методы добавления линий, станций и пересадок описаны в файле Metro.java).

Для примера была создана схема метро, показанная на рисунке 1.

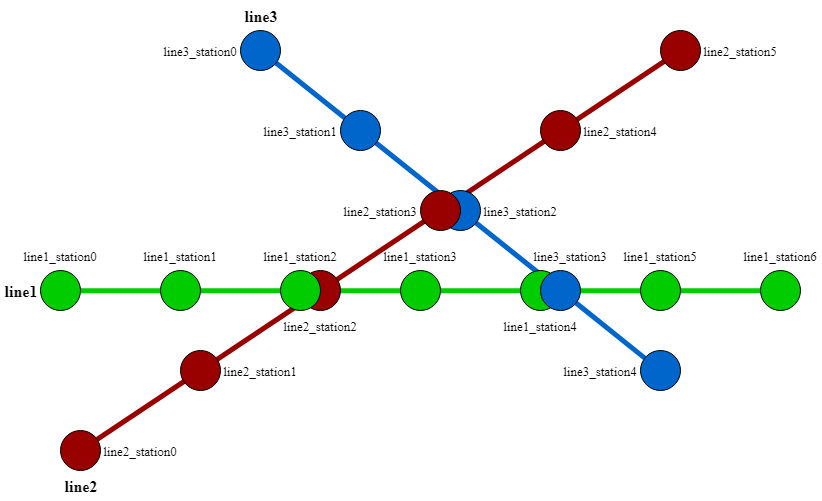


Рисунок 1 – Схема метро.

После создания графа, среда посылает всем созданным на данный момент агентам (метро и пассажиры) информацию о линиях и станциях. Агент-метро (файл metro.asl), получив данную информацию, создает станции и поезда в виде агентов (station.asl и train.asl) и отправляет всем агентам сигнал об открытии метро (убеждение metroIsOpen).

Агенты-поезда (файл train.asl), получив убеждение metroIsOpen, инициализируют свою начальную станцию и начинают движение.

Агенты-пассажиры (файл passenger.asl), получив убеждение metroIsOpen, инициализируют себе начальную и конечную станцию и запрашивают у среды маршрут пути (путь ищется с помощью Алгоритма Дейкстры, описанного в файлах Dijkstra.java и Node.java). Получив путь, пассажир получает направление и проверяет, находится ли он на станции пересадки – если нет, то он получает направление и ждет прибытия поезда на станцию, иначе делает пересадку и снова проверяет текущую станцию.

Приехав на агент-станцию (файл station.asl), поезд сообщает   
агентам-пассажирам, находящихся в данном поезде, о текущей станции. Если текущая станция является для пассажира конечной станцией или станцией, на которой нужно сделать пересадку, то пассажир выходит на данной станции. При этом те пассажиры, которые были на станции, садятся в поезд и поезд едет дальше. Доезжая до последней станции на линии, поезд меняет свое направление и продолжает движение.

Доехав до своей конечной станции, пассажир удаляет себя из мультиагентной системы. Программа считается успешно выполненной, если все пассажиры доехали до своей конечной станции или, иначе говоря, если все пассажиры удалились из системы.

1. Листинг программы.

|  |
| --- |
| metro.asl |
| /\* Initial beliefs and rules \*/  /\* Initial goals \*/  !initStationsAndTrains.  /\* Plans \*/  //Создание агентов линий и агентов поездов,  //основываясь на количестве линий и станций,  //полученных от среды  +!initStationsAndTrains <-  .findall(LineName, line(LineName, \_), L);  for(.member(Line, L)) {  ?line(Line, LineStations);  for(.member(Station, LineStations)) {  .nth(I, LineStations, Station);  .concat(Line, "\_station", I, StationName);  .create\_agent(StationName, "station.asl");  .send(Station, tell, mainLine(Line));  .concat(Line, "\_train", I, TrainName);  .create\_agent(TrainName, "train.asl");  .send(TrainName, tell, mainLine(Line));  }  };  .broadcast(tell, metroIsOpen). |

|  |
| --- |
| train.asl |
| /\* Initial beliefs and rules \*/  p\_list([]).  /\* Initial goals \*/  /\* Plans \*/  +metroIsOpen <-  !getStartStation;  !go.  //Получение начального направления и начальной станции  +!getStartStation <-  ?mainLine(MainLine);  ?line(MainLine, StationList);  D = math.round(math.random(1));  +direction(D);  .length(StationList, StationListLength);  StationPosition = math.round(math.random(StationListLength - 1));  .nth(StationPosition, StationList, Station);  .send(Station, askOne, train(D), A);  //Если на полученной станции уже есть поезд,  //то получить станцию заново  if (A \== false) {  !getStartStation;  } else {  -+currentStation(Station);  }.  //Здесь описывается движение поезда по направлению 0 или 1  //0 - от "меньшей" станции к "большей" (например, от station0 к station1)  //1 - от "большей" станции к "меньшей" (например, от station1 к station0)  +!go <-  ?currentStation(CurrentStation);  ?direction(D);  //Удаление из базы убеждения текущей станции данный поезд  .send(CurrentStation, untell, train(D));  ?mainLine(MainLine);  ?line(MainLine, StationList);  .nth(StationPosition, StationList, CurrentStation);  .length(StationList, StationListLength);  //Если поезд приезжает на конечную станцию,  //то сменить направление  if(D == 0) {  NextStationPosition = StationPosition + 1;  if(NextStationPosition == StationListLength) {  .nth(NextStationPosition - 1, StationList, NextStation);  -+direction(1);  !checkNextTrain(1, NextStation);  } else {  .nth(NextStationPosition, StationList, NextStation);  !checkNextTrain(0, NextStation);  }  } elif(D == 1){  NextStationPosition = StationPosition - 1;  if(NextStationPosition < 0) {  .nth(0, StationList, NextStation);  -+direction(0);  !checkNextTrain(0, NextStation);  } else {  .nth(NextStationPosition, StationList, NextStation);  !checkNextTrain(1, NextStation);  }  }.  //Проверка наличия поезда на следующей линии  +!checkNextTrain(D, NextStation) <-  .send(NextStation, askOne, train(D), A);  -+currentStation(NextStation);  ?p\_list(PL);  //Добавление в базу убеждения следующей станции данный поезд  .send(NextStation, tell, train(D));  //Прибытие на следующую станцию  .send(NextStation, achieve, arrived(D, PL)).  //Обновление списка пассажиров  +!updatePassengers(NewPassengersOnTrain)[source(Station)] <-  .abolish(p\_list(\_));  +p\_list(NewPassengersOnTrain). |

|  |
| --- |
| station.asl |
| /\* Initial beliefs and rules \*/  /\* Initial goals \*/  /\* Plans \*/  +!arrived(D, PassengersOnTrain)[source(T)] <-  ?mainLine(MainLine);  .my\_name(StationName);  //Получение списка пассажиров на станции  .findall(Name, passenger(D)[source(Name)], PassengersOnStation);  //Удаление всех пассажиров из базы убеждения данной станции  .abolish(passenger(D));  //Если пассажир приехал,  //то удалить его из списка поезда  for(.member(PT, PassengersOnTrain)) {  .send(PT, achieve, checkStation);  .wait({+passengerArrived[source(PT)]}, 100, Time);  if(Time < 100) {  .send(T, askOne, p\_list(CurrentPassengers), p\_list(CurrentPassengers));  .delete(PT, CurrentPassengers, NewPassengersOnTrain);  .send(T, achieve, updatePassengers(NewPassengersOnTrain));  .abolish(passengerArrived[source(PT)]);  }  }  //Конкатенация получившегося списка поезда со списком станции и  //отправление нового списка поезду  .send(T, askOne, p\_list(PassengersAfterArrival), p\_list(PassengersAfterArrival));  .concat(PassengersAfterArrival, PassengersOnStation, NewPassengersOnTrain);  .send(T, achieve, updatePassengers(NewPassengersOnTrain));  //Вывод на экран только тех поездов, у которых имеется непустой список пассажиров  .concat(PassengersOnTrain, NewPassengersOnTrain, List);  if(.length(List, Length) & Length > 0) {  .print("----- ", T, "(direction ", D, ") ----- arrived with ", PassengersOnTrain, ", departed with ", NewPassengersOnTrain);  }  //Задержка для имитации остановки поезда на станции  .wait(1000);  //Отправка поезду сигнала о продолжении движения  .send(T, achieve, go). |

|  |
| --- |
| passenger.asl |
| /\* Initial beliefs and rules \*/  /\* Initial goals \*/  /\* Plans \*/  +metroIsOpen <-  //Получение информации о количестве линий в метро  .count(line(\_, \_), LN);  //Инициализация начальной станции и линии  !initStation(LN, LB, B);  +currentLine(LB);  +currentStation(B);  //Инициализация конечной станции  !initStation(LN, LE, E);  //Получение пути от среды по алгоритму Дейкстры  getPath(B, E);  ?path(Path);  .print("My path is ", Path);  !checkStation.  +!initStation(LN, S\_line, S\_name) <-  //Выборка линии  L\_rand = math.round(math.random(LN - 1)) + 1;  .concat(line, L\_rand, L\_stringName);  .term2string(S\_lineTemp, L\_stringName);  //Выборка станции на линии  ?line(S\_lineTemp, S\_list);  .length(S\_list, S\_listLength);  S\_posTemp = math.round(math.random(S\_listLength - 1));  .nth(S\_posTemp, S\_list, S\_nameTemp);  //Если конечная станция будет равна стартовой,  //то провести инициализацию заново  if(currentStation(C) & C == S\_nameTemp) {  !initStation(LN, S\_line, S\_name);  } else {  S\_line = S\_lineTemp;  S\_name = S\_nameTemp;  }.  //Получение направления  +!getDirection([Current, Next|\_]) <-  ?currentStation(CurrentStation);  ?currentLine(CurrentLine);  ?line(CurrentLine, LineStations);  //Если следующая станция находится на текущей линии, то  //получаем номера текущей и следующей станции и сравниваем их  if(.member(Next, LineStations)) {  .concat(CurrentLine, "\_station", LineX\_station);  .delete(LineX\_station, CurrentStation, CurrentStringNumber);  .term2string(CurrentNumber, CurrentStringNumber);  .delete(LineX\_station, Next, NextStringNumber);  .term2string(NextNumber, NextStringNumber);  if(CurrentNumber < NextNumber) {  -+direction(0);  } else {  -+direction(1);  };  ?direction(D);  .my\_name(Name);  .send(CurrentStation, tell, passenger(D))  //Если следующая станция находится на другой линии, то нужно сделать пересадку  } else {  ?path(Path);  !transfer(Next, Path);  }.  //Пересадка  +!transfer(Next, [\_|PathAfterTransfer]) <-  //Заменяем текущую станцию и линию на следующие  .findall(LineName, line(LineName, \_), Lines);  for(.member(NextLine, Lines)) {  ?line(NextLine, Stations);  if(.member(Next, Stations)) {  -+currentLine(NextLine);  }  }  -+currentStation(Next);  -+path(PathAfterTransfer)[source(percept)];  //Если в списке пути после пересадки еще есть станции, то ищем направление  if(.length(PathAfterTransfer, PathLength) & PathLength > 1) {  !getDirection(PathAfterTransfer);  //Если нет - пассажир приехал  } else {  .print("!!!!!!!!!!!!!!!!!!!! I arrived to my destination (", Next, ") after transfer !!!!!!!!!!!!!!!!!!!!");  .my\_name(N);  .kill\_agent(N);  }.  //Если начальная и следующая станции находятся на разных линиях,  //то сделать пересадку  +!checkStation[source(self)] :  currentStation(Station) &  path([Station, TransferStation|Other]) & currentLine(CurrentLine) &  line(CurrentLine, LineStations) &  not .member(TransferStation, LineStations) <-  -+currentStation(Station);  ?path(Path);  ?currentStation(Curr);  .print("---------- Was on ", Station, ", making transfer to ", TransferStation, " ----------");  .send(Station, tell, passengerArrived);  !getDirection(Path).  //Если начальная и следующая станции находятся на одной линии,  //то найти направление  +!checkStation[source(self)] <-  ?path(Path);  !getDirection(Path).  //Если в списке пути больше нет станций,  //то пассажир приехал на конечную станцию  +!checkStation[source(Station)] : path([\_, Station|[]]) <-  -+currentStation(Station);  .send(Station, tell, passengerArrived);  .print("!!!!!!!!!!!!!!!!!!!! I arrived to my destination (", Station, ") !!!!!!!!!!!!!!!!!!!!");  .my\_name(N);  .kill\_agent(N).  //Если текущая и следующая станции находятся на разных линиях,  //то пассажир приехал на станцию пересадки  +!checkStation[source(Station)] :  path([\_, Station, TransferStation|Other]) & currentLine(CurrentLine) &  line(CurrentLine, LineStations) &  not .member(TransferStation, LineStations) <-  -+currentStation(Station);  ?path([\_|Path]);  -+path(Path)[source(percept)];  .send(Station, tell, passengerArrived);  ?currentStation(Curr);  .print("---------- Was on ", Station, ", making transfer to ", TransferStation, " ----------");  !transfer(TransferStation, Path).  //Если текущая и следующая станции находятся на одной линии,  //то пассажир едет дальше  +!checkStation[source(Station)]: path([\_, Station|\_]) <-  -+currentStation(Station);  ?path([\_|Path]);  -+path(Path)[source(percept)];  .print("Not my station yet.");. |

|  |
| --- |
| MetroEnv.java |
| import java.util.\*;  import jason.asSyntax.\*;  import jason.environment.\*;  import jason.runtime.\*;  import java.util.logging.\*;  public class MetroEnv extends jason.environment.TimeSteppedEnvironment {  private Logger logger = Logger.getLogger("metro.mas2j." + MetroEnv.class.getName());  private Metro metro = new Metro();    /\*\* Called before the MAS execution with the args informed in .mas2j \*/  @Override  public void init(String[] args) {  super.init(new String[] { "500" } );  setOverActionsPolicy(OverActionsPolicy.ignoreSecond);  LinkedList<String> line1 = new LinkedList<>();  LinkedList<String> line2 = new LinkedList<>();  LinkedList<String> line3 = new LinkedList<>();  try {  for (int i = 0; i <= 6; i++) {  String stationName = "line1\_station" + i;  line1.add(stationName);  }  for (int i = 0; i <= 5; i++) {  String stationName = "line2\_station" + i;  line2.add(stationName);  }  for (int i = 0; i <= 4; i++) {  String stationName = "line3\_station" + i;  line3.add(stationName);  }  } catch(Exception e){  logger.info("Failed to init stations!");  };  LinkedList<String> transfer1 = new LinkedList<>();  transfer1.add("line1\_station2");  transfer1.add("line2\_station2");  LinkedList<String> transfer2 = new LinkedList<>();  transfer2.add("line2\_station3");  transfer2.add("line3\_station2");  LinkedList<String> transfer3 = new LinkedList<>();  transfer3.add("line1\_station4");  transfer3.add("line3\_station3");    metro.addLine(line1);  metro.addLine(line2);  metro.addLine(line3);  metro.addTransfer(transfer1);  metro.addTransfer(transfer2);  metro.addTransfer(transfer3);  metro.calculateFastestPath("line1\_station6", "line2\_station5");  metro.calculateFastestPath("line2\_station3", "line1\_station0");    addPercept(Literal.parseLiteral("line(line1, " + line1.toString() + ")"));  addPercept(Literal.parseLiteral("line(line2, " + line2.toString() + ")"));  addPercept(Literal.parseLiteral("line(line3, " + line3.toString() + ")"));  }  @Override  public boolean executeAction(String agName, Structure action) {  if (action.getFunctor().equals("getPath")) {  List<String> path = metro.calculateFastestPath(action.getTerms().get(0).toString(),  action.getTerms().get(1).toString());  addPercept(agName, Literal.parseLiteral("path(" + path.toString() + ")"));  }  return true; // the action was executed with success  }    /\*\* Called before the end of MAS execution \*/  @Override  public void stop() {  super.stop();  }  } |

|  |
| --- |
| Metro.java |
| import java.util.\*;  class Metro {  private HashSet<Node> stations = new HashSet<>();  private HashMap<String, Node> stationsMap = new HashMap<>();  private void populateStationsMap(LinkedList<String> line){  for (String stationName : line) {  if (!stationsMap.containsKey(stationName)){  Node currentStation = new Node(stationName);  stationsMap.put(stationName, currentStation);  }  }  }  public void addLine(LinkedList<String> line) {  populateStationsMap(line);  for (String stationName : line) {  addAdjacentStations(line, getStation(stationName));  stations.add(getStation(stationName));  }  }  public void addTransfer(LinkedList<String> line) {  populateStationsMap(line);  for (String station : line) {  for (String anotherStation : line) {  if (!anotherStation.equals(station)){  getStation(station).addDestination(getStation(anotherStation),1);  }  }  stations.add(getStation(station));  }  }  private void addAdjacentStations(LinkedList<String> line, Node currentStation){  int index = line.indexOf(currentStation.getName());  if (index < line.size() - 1)  currentStation.addDestination(getStation(line.get(index + 1)), 1);  if (index - 1 >= 0)  currentStation.addDestination(getStation(line.get(index - 1)), 1);  }  public Node getStation(String stationName){  return stationsMap.get(stationName);  }  public synchronized List<String> calculateFastestPath(String start, String end){  stations.forEach(Node::reset);  Dijkstra.calculateShortestPathFromSource(getStation(start));  List<String> pathStations = new LinkedList<>();  for (Node station : getStation(end).getShortestPath()) {  pathStations.add(station.getName());  }  pathStations.add(end);  return pathStations;  }  } |

|  |
| --- |
| Dijkstra.java |
| import java.util.\*;  class Dijkstra {  public static void calculateShortestPathFromSource(Node source) {  Set<Node> settledNodes = new HashSet<>();  Set<Node> unsettledNodes = new HashSet<>();    source.setDistance(0);  unsettledNodes.add(source);  while (unsettledNodes.size() != 0) {  Node currentNode = getLowestDistanceNode(unsettledNodes);  unsettledNodes.remove(currentNode);  for (Map.Entry<Node, Integer> adjacencyPair :  currentNode.getAdjacentNodes().entrySet()) {  Node adjacentNode = adjacencyPair.getKey();  Integer edgeWeight = adjacencyPair.getValue();  if (!settledNodes.contains(adjacentNode)) {  calculateMinimumDistance(adjacentNode, edgeWeight, currentNode);  unsettledNodes.add(adjacentNode);  }  }  settledNodes.add(currentNode);  }  }  private static Node getLowestDistanceNode(Set<Node> unsettledNodes) {  Node lowestDistanceNode = null;  int lowestDistance = Integer.MAX\_VALUE;  for (Node node : unsettledNodes) {  int nodeDistance = node.getDistance();  if (nodeDistance < lowestDistance) {  lowestDistance = nodeDistance;  lowestDistanceNode = node;  }  }  return lowestDistanceNode;  }  private static void calculateMinimumDistance(Node evaluationNode, Integer edgeWeigh, Node sourceNode) {  Integer sourceDistance = sourceNode.getDistance();  if (sourceDistance + edgeWeigh < evaluationNode.getDistance()) {  evaluationNode.setDistance(sourceDistance + edgeWeigh);  LinkedList<Node> shortestPath = new LinkedList<>(sourceNode.getShortestPath());  shortestPath.add(sourceNode);  evaluationNode.setShortestPath(shortestPath);  }  }  } |

|  |
| --- |
| Node.java |
| import java.util.\*;  class Node {  private String name;  private List<Node> shortestPath = new LinkedList<>();  private Integer distance = Integer.MAX\_VALUE;  private Map<Node, Integer> adjacentNodes = new HashMap<>();    public void addDestination(Node destination, int distance) {  adjacentNodes.put(destination, distance);  }  public void reset(){  this.shortestPath.clear();  this.distance = Integer.MAX\_VALUE;  }  public Node(String name) {  this.name = name;  }  public void setDistance(Integer distance) {  this.distance = distance;  }  public Integer getDistance() {  return distance;  }  public void setShortestPath(List<Node> shortestPath) {  this.shortestPath = shortestPath;  }  public List<Node> getShortestPath() {  return shortestPath;  }  public Map<Node, Integer> getAdjacentNodes() {  return adjacentNodes;  }  public String getName() {  return name;  }  } |

1. Результат работы программы.

На рисунке 2 показан пример работы программы с 5-ю пассажирами. Так как все они доехали до своей конечной станции, то в отладчике их не видно – они удалились из мультиагентной системы.

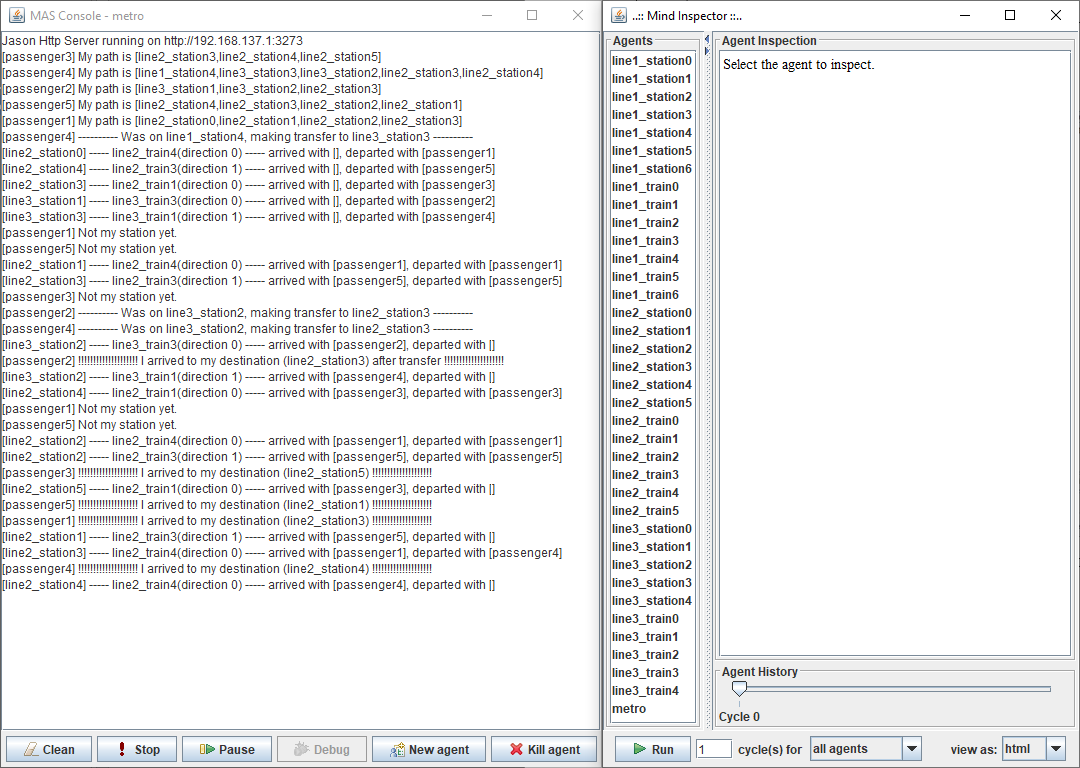


Рисунок 2 – Пример работы программы с 5-ю пассажирами.