Programmation Avance : Devoir Maison Tron

Licence Générale en Informatique

UFA Charles de Foucault

Etudiant : Nicolas ANTUNES

**Pseudo CodingGame :** Astuna

**Depot GitHub :**

<https://github.com/Nestuna/LG_Prog_Avancee/commits/master/CodingGame_Tron>

**Code :** (à partir de la page suivante)

import java.util.\*;

import javax.sound.midi.SysexMessage;

/\*\*

\* Auto-generated code below aims at helping you parse

\* the standard input according to the problem statement.

\*\*/

class Robot {

private Boolean isOpponent;

private Position initPosition;

private Position currentPosition;

Robot (int x0, int y0, int x1, int y1) {

this.initPosition = new Position(x0, y0) ;

this.currentPosition = new Position(x1, y1);

}

// ------------------------------ Coordonates

public Position getInitPosition () {

return this.initPosition;

}

public Position getPosition () {

return this.currentPosition;

}

public void setX (int x) {

this.currentPosition.setX(x);

}

public void setY (int y) {

this.currentPosition.setY(y);

}

public void setPosition (int x1, int y1) {

this.currentPosition.setX(x1);

this.currentPosition.setY(y1);

}

// ------------------------------ Distances

public int getDistanceFromX (int x) {

return Math.abs(this.currentPosition.getX() - x);

}

public int getDistanceFromY (int y) {

return Math.abs(this.currentPosition.getY() - y);

}

public int getDistanceFromInitX () {

return Math.abs(this.initPosition.getX() - this.currentPosition.getX());

}

public int getDistanceFromInitY () {

return Math.abs(this.initPosition.getY() - this.currentPosition.getY());

}

@Override

public String toString() {

String robotStatus = isOpponent ? "Oppennent" : "Player";

return robotStatus + ":\n" + "\t Init = " + this.initPosition.toString() + "\n\t Current = " + this.currentPosition.toString();

}

}

class OpponentRobot extends Robot {

OpponentRobot (int x0, int y0, int x1, int y1) {

super (x0, y0, x1, y1);

}

}

class PlayerRobot extends Robot {

PlayerRobot (int x0, int y0, int x1, int y1) {

super(x0, y0, x1, y1);

}

}

class Grid {

private int x;

private int y;

private int[][] map;

Grid () {

this.x = 30;

this.y = 20;

createMap(this.x, this.y);

}

Grid (int x, int y) {

this.x = x;

this.y = y;

createMap(this.x, this.y);

}

Grid (Position position) {

this.x = position.getX();

this.y = position.getY();

createMap(this.x, this.y);

}

public void createMap (int x, int y) {

this.map = new int[x][y];

for (int i=0; i < x; i++) {

for (int j=0; j < y; j++) {

this.map[i][j] = 0;

}

}

}

public ArrayList<Position> getLinesPositions() {

ArrayList<Position> positions = new ArrayList<Position>();

for(int x = 0; x < this.x; x++) {

for(int y = 0; y < this.y; y++) {

if (this.map[x][y] > 0) {

Position pos = new Position(x, y);

positions.add(pos);

}

}

}

return positions;

}

public ArrayList<ArrayList<Integer>> getRobotLinePositions(int robotId) {

ArrayList<ArrayList<Integer>> positions = new ArrayList<ArrayList<Integer>>();

for(int x = 0; x < this.x; x++) {

for(int y = 0; y < this.y; y++) {

if (this.map[x][y] == robotId) {

ArrayList<Integer> pos = new ArrayList<Integer>(Arrays.asList(x, y));

positions.add(pos);

}

}

}

return positions;

}

public void setLinePosition (int x, int y, int player) {

// player = 1 if player, 2 if opponent

this.map[x][y] = player;

}

public void setLinePosition (Position position, int player) {

x = position.getX();

y = position.getY();

// player = 1 if player, 2 if opponent

this.map[x][y] = player;

}

public Boolean isWall(int x, int y) {

if(x < 0 || x >= this.x || y < 0 || y >= this.y)

return true;

return false;

}

public Boolean isWall(Position position) {

int x = position.getX();

int y = position.getY();

if(x < 0 || x >= this.x || y < 0 || y >= this.y ) {

return true;

}

return false;

}

public Boolean isLine(int x, int y) {

if (this.map[x][y] > 0) return true;

return false;

}

public Boolean isLine(Position position) {

int x = position.getX();

int y = position.getY();

if (this.map[x][y] > 0) {

return true;

}

return false;

}

@Override

public String toString() {

String mapStr = "";

for(int j = 0 ; j < this.y ; j++) {

for (int i = 0 ; i < this.x ; i++)

mapStr += this.map[i][j];

mapStr += "\n";

}

return mapStr;

}

}

class PathFinder {

public Graph graph;

public Grid map;

private LinkedList<Node> visitedNodes;

private Node start;

private LinkedList<String> directions;

PathFinder (Position startPosition, Grid map) {

this.map = map;

this.graph = initGraph();

this.start = this.graph.getNode(startPosition);

}

private Graph initGraph() {

Graph graph = new Graph();

for (int x = 0; x < Graph.X; x++) {

for (int y = 0; y < Graph.Y; y++) {

graph.addNode(new Position(x, y));

}

}

for (Node node : graph.getNodesList().values()) {

for (Position nextPosition : node.getPosition().nextPositions().values()) {

Node nextNode = graph.getNode(nextPosition);

if (nextNode != null) {

node.connect(nextNode);

}

}

}

return graph;

}

@Override

public String toString() {

return this.graph.toString();

}

public boolean isCorrectMove(Position position) {

if (this.map.isWall(position)) return false;

else if (this.map.isLine(position)) return false;

else return true;

}

public String nextDirection(LinkedList<Node> path) {

String nextDirection = "DOWN";

Node prevNode = path.removeFirst();

if (!path.isEmpty()) {

Node nextNode = path.getFirst();

Position nextPosition = nextNode.getPosition();

Position prevPosition = prevNode.getPosition();

for (String direction : prevPosition.nextPositions().keySet()) {

if (nextPosition.equals(prevPosition.nextPositions().get(direction))) {

nextDirection = direction.toUpperCase();

}

}

}

return nextDirection;

}

public String findShortestPath (Position endPosition) {

LinkedList<Node> nodes = findNodesForShortestPath(endPosition);

graph = connectPathNodes(nodes);

LinkedList<Node> path = findShortestPath(graph);

return nextDirection(path);

}

public LinkedList<Node> findShortestPath (Graph graphPath) {

// Le dernier noeud est la destination

// Il suffit de remonter le BFS à l'envers en cherchant le voisin dans la liste

// à partir de la destination

LinkedList<Node> nodesTree = new LinkedList<>(graphPath.getNodesList().values());

LinkedList<Node> reversePath = new LinkedList<>();

Node lastNode = nodesTree.getLast();

reversePath = findShortestPath(reversePath, nodesTree, lastNode);

reversePath.addFirst(lastNode);

LinkedList<Node> path = new LinkedList<>();

while (!reversePath.isEmpty()) {

path.add(reversePath.getLast());

reversePath.removeLast();

}

return path;

}

public LinkedList<Node> findShortestPath(LinkedList<Node> path, LinkedList<Node> nodesTree, Node lastNode) {

if (lastNode.getPosition() != this.start.getPosition()) {

for (Node parentNode : nodesTree) {

for (Node childNode : parentNode.getNeighbors()) {

if (childNode == lastNode) {

lastNode = parentNode;

path.add(lastNode);

return findShortestPath(path, nodesTree, lastNode);

}

}

}

}

return path;

}

public LinkedList<Node> findNodesForShortestPath (Position endPosition) {

LinkedList<Node> queue = new LinkedList<>();

this.visitedNodes = new LinkedList<>();

Node end = this.graph.getNode(endPosition);

queue.add(this.start);

visitedNodes.add(this.start);

while(!this.visitedNodes.contains(end) && !queue.isEmpty()) {

Node nextNode = queue.getFirst();

queue.pop();

for (Node node: nextNode.getNeighbors()) {

if (!this.visitedNodes.contains(node) && !this.visitedNodes.contains(end) && isCorrectMove(node.getPosition())) {

queue.add(node);

this.visitedNodes.add(node);

}

}

}

return this.visitedNodes;

}

Graph connectPathNodes (LinkedList<Node> nodesList) {

// On connecte les noeuds en sens unique pour avoir un chemin

Graph pathGraph = new Graph();

for (Node node : nodesList) {

Position position = node.getPosition();

pathGraph.addNode(position);

}

HashSet<Node> connected = new HashSet<>();

for (Node node : pathGraph.getNodesList().values()) {

for (Position nextPosition : node.getPosition().nextPositions().values()) {

Node nextNode = pathGraph.getNode(nextPosition);

if (!connected.contains(nextNode) && nextNode != null) {

node.addNeighbor(nextNode);

connected.add(nextNode);

}

}

}

return pathGraph;

}

}

class Graph {

final static int X = 30, Y = 20;

private LinkedHashMap<Position, Node> nodesList;

Graph () {

nodesList = new LinkedHashMap<>();

}

// ------------------------ Getters & Setters

public Node getNode (int x, int y) {

Position position = new Position(x, y);

return nodesList.get(position);

}

public Node getNode (Position position) {

return nodesList.get(position);

}

public LinkedHashMap<Position, Node> getNodesList () {

return nodesList;

}

public Node addNode (Position position) {

Node node = new Node(position);

nodesList.put(position, node);

return node;

}

// ------------------------ Methods

@Override

public String toString () {

return nodesList.toString();

}

}

class Node {

private Position position;

private HashSet<Node> neighbors;

private int distance;

Node (int x, int y) {

this.position = new Position(x, y);

}

Node (Position position) {

this.position = position;

this.neighbors = new HashSet<>();

}

Node (Position position, int distance) {

this.position = position;

this.distance = distance;

this.neighbors = new HashSet<>();

}

Node (Node other) {

this.position = other.position;

}

// ------------------------ Getters & Setters

public Position getPosition() {

return this.position;

}

public HashSet<Node> getNeighbors() {

return this.neighbors;

}

public void addNeighbor(Node node) {

this.neighbors.add(node);

}

public int getDistance() {

return this.distance;

}

public void setDistance(int distance) {

this.distance = distance;

}

// ------------------------ Methods

public void connect(Node node) {

if (node != this) {

this.neighbors.add(node);

node.neighbors.add(this);

}

}

// @Override

// public String toString() {

// return this.position.toString();

// }

public String toString () {

String str = this.getPosition().toString();

str += " : [ ";

for (Node neighbor : this.neighbors)

str += neighbor.getPosition().toString() + " ";

return str + "]\n";

}

}

class Position {

private int x;

private int y;

public HashMap<String, Position> nextPositions;

Position(int x, int y) {

this.x = x;

this.y = y;

}

@Override

public int hashCode()

{

return Graph.X \* x + y;

}

// Getters & Setters

public int getX () {

return this.x;

}

public int getY () {

return this.y;

}

public void setX (int x) {

this.x = x;

}

public void setY (int y) {

this.y = y;

}

public void setPosition (Position position) {

this.x = position.x;

this.y = position.y;

}

public HashMap<String, Position> nextPositions () {

this.nextPositions = new HashMap<>();

this.nextPositions.put("up", new Position(this.x, this.y - 1));

this.nextPositions.put("right", new Position(this.x + 1, this.y));

this.nextPositions.put("down", new Position(this.x, this.y + 1));

this.nextPositions.put("left", new Position(this.x - 1, this.y));

return this.nextPositions;

}

@Override

public String toString() {

return String.format("(%1$d , %2$d)", this.x, this.y);

}

@Override

public boolean equals(Object pos) {

Position other = (Position) pos;

return other.x == x && other.y == y;

}

}

class Player {

// Robots and map attributs

static Robot player = null;

static HashMap<Integer, Robot> opponentsList = new HashMap<Integer, Robot>();

static Grid map = new Grid();

// Game Main Functions

public static Boolean isCorrectMove(Position position) {

if (map.isWall(position)) return false;

else if (map.isLine(position)) return false;

else {

return true;

}

}

public static Position getNearestWall(Robot player) {

Position destination = null;

Position start = player.getPosition();

int lessX = -1;

int distanceX = 100;

int[] rows = {player.getDistanceFromX(0), player.getDistanceFromX(29)};

if (rows[0] < rows[1] && isCorrectMove(new Position(rows[0], start.getY()))) {

distanceX = rows[0];

lessX = 0;

} else if (rows[0] > rows[1] && isCorrectMove(new Position(rows[1], start.getY()))) {

distanceX = rows[1];

lessX = 29;

}

int lessY = -1;

int distanceY = 100;

int [] cols = {player.getDistanceFromY(0), player.getDistanceFromY(19)};

if (cols[0] < cols[1] && isCorrectMove(new Position(start.getY(), cols[0]))) {

distanceY = cols[0];

lessY = 0;

} else if (cols[0] > cols[1] && isCorrectMove(new Position(start.getY(), cols[1]))) {

distanceY = cols[1];

lessY = 19;

}

if (distanceX < distanceY) {

destination = new Position(lessX, start.getY());

} else if (distanceX > distanceY) {

destination = new Position(start.getX(), lessY);

}

if (destination != null && destination.equals(start)) destination = null;

return destination;

}

public static Position getNearestCorner(Robot player) {

Position destination = null;

Position start = player.getPosition();

if (start.getX() == 0 || start.getX() == 29) {

int lessY = -1;

int distanceX = -1;

int[] rows = {player.getDistanceFromY(0), player.getDistanceFromY(19)};

if (rows[0] < rows[1] && isCorrectMove(new Position(rows[0], start.getY()))) {

distanceX = rows[0];

lessY = 0;

} else if (rows[0] > rows[1] && isCorrectMove(new Position(rows[1], start.getY()))) {

distanceX = rows[1];

lessY = 19;

}

if (distanceX >= 0) destination = new Position(start.getX(), lessY);

}

else if (start.getY() == 0 || start.getY() == 19) {

int [] cols = {player.getDistanceFromX(0), player.getDistanceFromX(29)};

int lessX = -1;

int distanceY = -1;

if (cols[0] < cols[1] && isCorrectMove(new Position(start.getY(), cols[0]))) {

distanceY = cols[0];

lessX = 0;

} else if (cols[0] > cols[1] && isCorrectMove(new Position(start.getY(), cols[1]))) {

distanceY = cols[1];

lessX = 29;

}

if (distanceY >= 0) destination = new Position(lessX, start.getY());

}

if (destination != null && (destination.equals(start) || !isCorrectMove(destination))) destination = null;

return destination;

}

public static String moveToMake(Robot player, HashMap<Integer, Robot> opponentsList, Grid map) {

Position start = player.getPosition();

Position destination = null;

if (start.getX() == 0 || start.getX() == 29 || start.getY() == 0 || start.getY() == 19 ) {

destination = getNearestCorner(player);

if (destination == null) {

destination = getNearestWall(player);

}

} else {

destination = getNearestWall(player);

}

if (destination == null) {

String[] directions = { "UP", "DOWN" , "LEFT", "RIGHT"};

destination = new Position(-1,-1);

int j = 0;

while (!isCorrectMove(destination) && j < directions.length) {

destination = start.nextPositions().get(directions[j].toLowerCase());

j++;

}

}

String nextDirection;

PathFinder pathFinder = new PathFinder(start, map);

System.err.println("Destination : " + destination);

nextDirection = pathFinder.findShortestPath(destination);

return nextDirection;

}

public static void printPositionsInGame(Robot player, HashMap<Integer, Robot> opponents) {

System.err.println("Player : " + player.getPosition());

for (Integer opponent : opponents.keySet()) {

System.err.println("Opponent " + opponent + " : " + opponents.get(opponent).getPosition());

}

}

public static void main(String args[]) {

Scanner in = new Scanner(System.in);

// game loop

while (true) {

int N = in.nextInt(); // total number of players (2 to 4).

int P = in.nextInt(); // your player number (0 to 3).

for (int i = 0; i < N; i++) {

int X0 = in.nextInt(); // starting X coordinate of lightcycle (or -1)

int Y0 = in.nextInt(); // starting Y coordinate of lightcycle (or -1)

int X1 = in.nextInt(); // starting X coordinate of lightcycle (can be the same as X0 if you play before this player)

int Y1 = in.nextInt(); // starting Y coordinate of lightcycle (can be the same as Y0 if you play before this player)

if (i == P) {

if (player == null) {

player = new PlayerRobot(X0,Y0,X1,Y1);

map.setLinePosition(X0, Y0, 1);

} else {

player.setPosition(X1,Y1);

}

map.setLinePosition(X1, Y1, 1);

} else {

if (opponentsList.size() < N - 1) {

Robot opponent = new OpponentRobot(X0,Y0,X1,Y1);

opponentsList.put(i, opponent);

map.setLinePosition(X0, Y0, 2);

} else {

Robot opponent = opponentsList.get(i);

opponent.setPosition(X1, Y1);

opponentsList.put(i, opponent);

}

map.setLinePosition(X1, Y1, 2);

}

}

// DEBUG

printPositionsInGame(player, opponentsList);

// ACTION

String move = moveToMake(player, opponentsList, map);

System.out.println(move);

}

}

}