Hard Code:

Request.*move*(10);

Request.*turn*("E");

Request.*move*(4);

Request.*refuel*();

Request.*move*(14);

Request.*turn*("N");

Request.*move*(11);

Request.*turn*("E");

Request.*move*(2);

Request.*finish*();

Code:

import org.json.simple.JSONArray;

import org.json.simple.JSONObject;

public class MainClass {

public static void main(String[] args) throws Exception {

JSONObject request = Request.createInstance();

//System.out.println(request);

System.out.println(Request.getData());

JSONArray asteroids = (JSONArray) Request.getData().get("Asteroids");

JSONObject jsonIns = Request.getInstance();

JSONObject roomXObj = (JSONObject) ((JSONObject) ((JSONObject)jsonIns.get("payload")).get("constants"))

.get("ROOM\_DIMENSIONS");

System.out.println (roomXObj);

long roomX = (long)roomXObj.get("X\_MAX");

long roomY = (long)roomXObj.get("Y\_MAX");

String [][] asterMap = new String [(int)roomX][(int)roomY];

//char [] dataArray = data.toCharArray();

}

}

JSONObject sta = (JSONObject) Request.*getData*().get("Station1");

int staX = (int) (long) sta.get("x");

int staY = (int) (long) sta.get("y");

astMap[staX][staY] = "S";

sta = (JSONObject) Request.*getData*().get("Station2");

staX = (int) (long) sta.get("x");

staY = (int) (long) sta.get("y");

astMap[staX][staY] = "S";

sta = (JSONObject) Request.*getData*().get("Mars");

staX = (int) (long) sta.get("x");

staY = (int) (long) sta.get("y");

astMap[staX][staY] = "M";

<https://rosettacode.org/wiki/Maze_solving#Java>

public static void mapPoints() {

for (int i = 0; i < asterMap.length; i++) {

for (int j = 0; j < asterMap.length; j++) {

System.out.print(asterMap[i][j]);

}

System.out.println("");

}

}

<https://medium.com/@manpreetsingh.16.11.87/shortest-path-in-a-2d-array-java-653921063884>

import org.json.simple.JSONArray;

import org.json.simple.JSONObject;

public class MainClass {

public static String[][] asterMap;

public static void main(String[] args) throws Exception {

JSONObject request = Request.createInstance();

// System.out.println(request);

System.out.println(Request.getData());

JSONArray asteroids = (JSONArray) Request.getData().get("Asteroids");

JSONObject jsonIns = Request.getInstance();

JSONObject roomXObj = (JSONObject) ((JSONObject) ((JSONObject) jsonIns.get("payload")).get("constants"))

.get("ROOM\_DIMENSIONS");

System.out.println(roomXObj);

long roomX = (long) roomXObj.get("X\_MAX");

long roomY = (long) roomXObj.get("Y\_MAX");

asterMap = new String[(int) roomX][(int) roomY];

// Empty Grid

for (int i = 0; i < asterMap.length; i++) {

for (int j = 0; j < asterMap.length; j++) {

asterMap[i][j] = " - ";

}

}

// Station1

JSONObject sta = (JSONObject) Request.getData().get("Station1");

int staX = (int) (long) sta.get("x");

int staY = (int) (long) sta.get("y");

asterMap[staX][staY] = " S ";

// Station2

sta = (JSONObject) Request.getData().get("Station2");

staX = (int) (long) sta.get("x");

staY = (int) (long) sta.get("y");

asterMap[staX][staY] = " S ";

// Mars

sta = (JSONObject) Request.getData().get("Mars");

int marX = (int) (long) sta.get("x");

int marY = (int) (long) sta.get("y");

asterMap[marX][marY] = " M ";

for (int i = 0; i < asteroids.size(); i++) {

JSONObject current = (JSONObject) asteroids.get(i);

int x = (int) (long) current.get("x");

int y = (int) (long) current.get("y");

asterMap[x][y] = " A ";

}

MainClass.mapPoints();

System.out.println("");

System.out.println("");

System.out.println("");

solveMazeRecur(0, 0, marX, marY, false);

MainClass.mapPoints();

}

public static boolean solveMazeRecur(int locX, int locY, int marX, int marY, boolean Mdown) {

//System.out.println (locX + ", " + locY);

if (locX == marX && locY == marY) {

asterMap[locX][locY] = " \* ";

return true;

} else {

// Right

if ((locX + 1) < 23 && !(asterMap[locX + 1][locY].equals(" A "))) {

boolean right = solveMazeRecur(locX + 1, locY, marX, marY, true);

if (right == true) {

asterMap[locX + 1][locY] = " \* ";

return true;

}

}

// Up

if ((locY + 1) < 23 && !asterMap[locX][locY + 1].equals(" A ")) {

boolean up = solveMazeRecur(locX, locY + 1, marX, marY, false);

if (up == true) {

asterMap[locX][locY + 1] = " \* ";

return true;

}

}

//Left

if ((locX - 1) >= 23 && !asterMap[locX - 1][locY].equals(" A ")) {

boolean left = solveMazeRecur(locX - 1, locY, marX, marY, true);

if (left == true) {

asterMap[locX - 1][locY] = " \* ";

return true;

}

}

//Down

if (Mdown == true && (locY - 1) >= 0 && !asterMap[locX][locY - 1].equals(" A ")) {

boolean down = solveMazeRecur(locX, locY - 1, marX, marY, true);

if (down == true) {

asterMap[locX][locY-1] = " \* ";

return true;

}

}

return false;

}

}

public static void mapPoints() {

for (int i = 0; i < asterMap.length; i++) {

for (int j = 0; j < asterMap.length; j++) {

System.out.print(asterMap[i][j]);

}

System.out.println("");

}

}

public static void Execute(int marX, int marY) {

boolean solved = false;

int x = 0, y = 0;

char d = 'N';

while (!solved) {

if ((x + 1) < 23 && asterMap[x + 1][y].equals(" \* ")) {

if (d == 'E') {

Request.move(1);

} else {

Request.turn("E");

d = 'E';

Request.move(1);

}

x++;

} else if ((x - 1) >= 0 && asterMap[x - 1][y].equals(" \* ")) {

if (d == 'W') {

Request.move(1);

} else {

Request.turn("W");

d = 'W';

Request.move(1);

}

x--;

} else if ((y + 1) < 23 && asterMap[x][y + 1].equals(" \* ")) {

if (d == 'N') {

Request.move(1);

} else {

Request.turn("N");

d = 'N';

Request.move(1);

}

y++;

} else if ((y - 1) >= 0 && asterMap[x + 1][y - 1].equals(" \* ")) {

if (d == 'S') {

Request.move(1);

} else {

d = 'S';

Request.turn("S");

Request.move(1);

}

y--;

}

}

if (marX == x && marY == y)

{

solved = true;

}

}

}