***Hints and Help for the Lab 2***

Add obstable(s)

Add(OSTABLE, 2,2);

Line 39

public boolean executeAction(String ag, Structure action) {

} else if (action.getFunctor().equals("move\_towards")) {

int x = (int)((NumberTerm)action.getTerm(0)).solve();

int y = (int)((NumberTerm)action.getTerm(1)).solve();

boolean cond = model.moveTowards(x,y);

updatePercepts();

return cond;

boolean moveTowards(int x, int y) throws Exception {

System.out.println("move towards");

Location r1 = getAgPos(0);

Location r2 = getAgPos(0); // sace the current loc

boolean cond = true;

System.out.println("Moving to "+x+" " + y);

System.out.flush();

if (r1.x < x)

r1.x++;

else if (r1.x > x)

r1.x--;

if (r1.y < y)

r1.y++;

else if (r1.y > y)

r1.y--;

if (hasObject(OBSTACLE,r1)) {

System.out.println("move failed " + r1.x+ " " +r1.y);

System.out.flush();

cond = false;

r1 = r2 ; // don't move on fail

}

setAgPos(0, r1);

setAgPos(1, getAgPos(1)); // just to draw it in the view

return cond;

}

Agent

/\* moveOneStep tries to move \*/

/\* and if it fails to move because of an obstacle it moves sideways \*/

/\* The first plan is just a check to see we are at the destination and removes the \*/

/\* guard terms q(x,moveOneStep) \*/

+!moveOneStep(l): at(L) <- .abolish(q(\_,moveOneStep)).

/\* Second plan, just do the normal move, add the term q(1,moveOneStep) \*/

/\* the plan executes as long as q(1,moveOneStep) has not been put in the beliefs \*/

+!moveOneStep(L) : not q(1,moveOneStep) <-

.print("step 1");

+q(1,moveOneStep);

?pos(L,X,Y);

move\_towards(X,Y);

.abolish(q(\_,moveOneStep)).

/\* Plan 2, change the x coordinate to move around the obstacle \*/

/\* uses guard q(2,moveOneStep) \*/

/\* a better version of this would look at the current r1 position and use it as the \*/

/\* the coordinate to modify for the the call to move\_towards \*/

+!moveOneStep(L) : not q(2,moveOneStep) <-

.print("step 2 ");

+q(2,moveOneStep);

?pos(L,X,Y);

move\_towards(X+1,Y);

.abolish(q(\_,moveOneStep)).

/\* this catches the failure of moveOneStep (note the -! syntax) \*/

/\* it asserts the gaol again to retry to achieve the move \*/

-!moveOneStep(L) : true <-

.print("do fail");

!moveOneStep(L).

// mars robot 1

/\* Initial beliefs \*/

at(P) :- pos(P,X,Y) & pos(r1,X,Y).

/\* Initial goal \*/

!check(slots).

/\* Plans \*/

+!check(slots) : not garbage(r1)

<- next(slot);

!check(slots).

+!check(slots).

@lg[atomic]

+garbage(r1) : not .desire(carry\_to(r2))

<- !carry\_to(r2).

+!carry\_to(R)

<- // remember where to go back

?pos(r1,X,Y);

-+pos(last,X,Y);

// carry garbage to r2

!take(garb,R);

// goes back and continue to check

!at(last);

!check(slots).

+!take(S,L) : true

<- !ensure\_pick(S);

!at(L);

drop(S).

+!ensure\_pick(S) : garbage(r1)

<- pick(garb);

!ensure\_pick(S).

+!ensure\_pick(\_).

+!at(L) : at(L).

+!at(L) <- ?pos(L,X,Y);

.print("move towards call");

!moveOneStep(L); /\* changed from original \*/

!at(L).

/\* moveOneStep tries to move \*/

/\* and if it fails to move because of an obstacle it moves sideways \*/

/\* The first plan is just a check to see we are at the destination and removes the \*/

/\* guard terms q(x,moveOneStep) \*/

+!moveOneStep(l): at(L) <- .abolish(q(\_,moveOneStep)).

/\* Second plan, just do the normal move, add the term q(1,moveOneStep) \*/

/\* the plan executes as long as q(1,moveOneStep) has not been put in the beliefs \*/

+!moveOneStep(L) : not q(1,moveOneStep) <-

.print("step 1");

+q(1,moveOneStep);

?pos(L,X,Y);

move\_towards(X,Y);

.abolish(q(\_,moveOneStep)).

/\* Plan 2, change the x coordinate to move around the obstacle \*/

/\* uses guard q(2,moveOneStep) \*/

/\* a better version of this would look at the current r1 position and use it as the \*/

/\* the coordinate to modify for the the call to move\_towards \*/

+!moveOneStep(L) : not q(2,moveOneStep) <-

.print("step 2 ");

+q(2,moveOneStep);

?pos(L,X,Y);

move\_towards(X+1,Y);

.abolish(q(\_,moveOneStep)).

/\* this catches the failure of moveOneStep (note the -! syntax) \*/

/\* it asserts the gaol again to retry to achieve the move \*/

-!moveOneStep(L) : true <-

.print("do fail");

!moveOneStep(L).

import jason.asSyntax.\*;

import jason.environment.Environment;

import jason.environment.grid.GridWorldModel;

import jason.environment.grid.GridWorldView;

import jason.environment.grid.Location;

import java.awt.Color;

import java.awt.Font;

import java.awt.Graphics;

import java.util.Random;

import java.util.logging.Logger;

public class MarsEnv extends Environment {

public static final int GSize = 7; // grid size

public static final int GARB = 16; // garbage code in grid model

public static final Term ns = Literal.parseLiteral("next(slot)");

public static final Term pg = Literal.parseLiteral("pick(garb)");

public static final Term dg = Literal.parseLiteral("drop(garb)");

public static final Term bg = Literal.parseLiteral("burn(garb)");

public static final Literal g1 = Literal.parseLiteral("garbage(r1)");

public static final Literal g2 = Literal.parseLiteral("garbage(r2)");

static Logger logger = Logger.getLogger(MarsEnv.class.getName());

private MarsModel model;

private MarsView view;

@Override

public void init(String[] args) {

model = new MarsModel();

view = new MarsView(model);

model.setView(view);

updatePercepts();

}

@Override

public boolean executeAction(String ag, Structure action) {

logger.info(ag+" doing: "+ action);

try {

if (action.equals(ns)) {

model.nextSlot();

} else if (action.getFunctor().equals("move\_towards")) {

int x = (int)((NumberTerm)action.getTerm(0)).solve();

int y = (int)((NumberTerm)action.getTerm(1)).solve();

/\* changes here - save the return values from moveTowards \*/

/\* update the percepts and then return the boolean result \*/

boolean cond = model.moveTowards(x,y);

updatePercepts();

return cond;

} else if (action.equals(pg)) {

model.pickGarb();

} else if (action.equals(dg)) {

model.dropGarb();

} else if (action.equals(bg)) {

model.burnGarb();

} else {

return false;

}

} catch (Exception e) {

e.printStackTrace();

}

updatePercepts();

try {

Thread.sleep(200);

} catch (Exception e) {}

informAgsEnvironmentChanged();

return true;

}

/\*\* creates the agents perception based on the MarsModel \*/

void updatePercepts() {

clearPercepts();

Location r1Loc = model.getAgPos(0);

Location r2Loc = model.getAgPos(1);

Literal pos1 = Literal.parseLiteral("pos(r1," + r1Loc.x + "," + r1Loc.y + ")");

Literal pos2 = Literal.parseLiteral("pos(r2," + r2Loc.x + "," + r2Loc.y + ")");

addPercept(pos1);

addPercept(pos2);

if (model.hasObject(GARB, r1Loc)) {

addPercept(g1);

}

if (model.hasObject(GARB, r2Loc)) {

addPercept(g2);

}

}

class MarsModel extends GridWorldModel {

public static final int MErr = 2; // max error in pick garb

int nerr; // number of tries of pick garb

boolean r1HasGarb = false; // whether r1 is carrying garbage or not

Random random = new Random(System.currentTimeMillis());

private MarsModel() {

super(GSize, GSize, 2);

// initial location of agents

try {

setAgPos(0, 0, 0);

Location r2Loc = new Location(GSize/2, GSize/2);

setAgPos(1, r2Loc);

} catch (Exception e) {

e.printStackTrace();

}

// initial location of garbage

add(GARB, 3, 0);

add(GARB, GSize-1, 0);

add(GARB, 1, 2);

add(GARB, 0, GSize-2);

add(GARB, GSize-1, GSize-1);

/\* change - add an obstacle \*/

add(OBSTACLE,3,2);

}

void nextSlot() throws Exception {

Location r1 = getAgPos(0);

r1.x++;

if (r1.x == getWidth()) {

r1.x = 0;

r1.y++;

}

// finished searching the whole grid

if (r1.y == getHeight()) {

return;

}

setAgPos(0, r1);

setAgPos(1, getAgPos(1)); // just to draw it in the view

}

boolean moveTowards(int x, int y) throws Exception {

System.out.println("move towards");

Location r1 = getAgPos(0);

Location r2 = getAgPos(0); // sace the current loc

boolean cond = true; // set return value, change it for a fail

System.out.println("Moving to "+x+" " + y);

System.out.flush();

// code to compute the next location

if (r1.x < x)

r1.x++;

else if (r1.x > x)

r1.x--;

if (r1.y < y)

r1.y++;

else if (r1.y > y)

r1.y--;

// test for obstacle on target square

if (hasObject(OBSTACLE,r1)) {

System.out.println("move failed " + r1.x+ " " +r1.y);

System.out.flush();

cond = false; // mark failure

r1 = r2 ; // don't move on fail

}

setAgPos(0, r1);

setAgPos(1, getAgPos(1)); // just to draw it in the view

return cond;

}

void pickGarb() {

// r1 location has garbage

if (model.hasObject(GARB, getAgPos(0))) {

// sometimes the "picking" action doesn't work

// but never more than MErr times

if (random.nextBoolean() || nerr == MErr) {

remove(GARB, getAgPos(0));

nerr = 0;

r1HasGarb = true;

} else {

nerr++;

}

}

}

void dropGarb() {

if (r1HasGarb) {

r1HasGarb = false;

add(GARB, getAgPos(0));

}

}

void burnGarb() {

// r2 location has garbage

if (model.hasObject(GARB, getAgPos(1))) {

remove(GARB, getAgPos(1));

}

}

}

class MarsView extends GridWorldView {

public MarsView(MarsModel model) {

super(model, "Mars World", 600);

defaultFont = new Font("Arial", Font.BOLD, 18); // change default font

setVisible(true);

repaint();

}

/\*\* draw application objects \*/

@Override

public void draw(Graphics g, int x, int y, int object) {

switch (object) {

case MarsEnv.GARB: drawGarb(g, x, y); break;

}

}

@Override

public void drawAgent(Graphics g, int x, int y, Color c, int id) {

String label = "R"+(id+1);

c = Color.blue;

if (id == 0) {

c = Color.yellow;

if (((MarsModel)model).r1HasGarb) {

label += " - G";

c = Color.orange;

}

}

super.drawAgent(g, x, y, c, -1);

if (id == 0) {

g.setColor(Color.black);

} else {

g.setColor(Color.white);

}

super.drawString(g, x, y, defaultFont, label);

repaint();

}

public void drawGarb(Graphics g, int x, int y) {

super.drawObstacle(g, x, y);

g.setColor(Color.white);

drawString(g, x, y, defaultFont, "G");

}

}

}