Bernat Martínez Gómez

1 Lunar lockout world definition

Objects

The lunar lockout world consist of:

- A 5x5 matrix defined using **index** items (**i0, i1, i2, i3, i4**) and giving them relations from top left to bottom right in this case.
- 6 identical objects of the type **spaceship** identified by colour (**orange, green, blue purple, yellow, red**)

States

The state has domain constraints:

- Spaceships can only move if they are able to make contact with another one in their path.
- Spaceships cannot use the border of the matrix to stop, they would go to outer space.
- The goal is moving the red spaceship to the center (i2, i2)

The ontology used to represent the environment is the following:
To represent the movement matrix we relate the index items in two ways, due to the definition of the problem we don't need to know more relations between them:

- **less(idx0, idx3)** means that idx0 represents a smaller value
- **one-less(idx0, idx1)** means that idx1 is right next to idx0

Having already the fined the world we can define where on the world are the spaceships placed using: on(red, idx0, idx1). This means that the red spaceship is on the position (0,1)

Actions

The definition of the environment can seem quite easy but defining the actions becomes a bit more complex in this problem, luckily there is a certain relation in the idea between them.

The parameter definition is different between vertical and horizontal movements but it follows a pattern. The red rectangle is the spaceship we want to move actual position, the green rectangle is the spaceship that we use as reference, and the blue rectangle is the position we want to move the spaceship to.
Note that in vertical movement the x value is the same for all of them and in the horizontal movement the y value is the one being shared.

Vertical movement parameters

Horizontal Movement Parameters

This is how the go up operation would work explained, the others work in a similar fashion but with swapped parameters/conditions depending on the type and direction of the movement:

go-up(sa spaceship, xab idx, ya idx, sb spaceship, yb idx, yo idx)
preconditions:
and(
on(sa xab ya) the spaceship we want to move is on its position
on(sb xab yb) the objective spaceship its in it position
one-less(yb yo) the objective spaceship is one value less in the y axis from the objective position (note that the idx values in the y axis start upwards)
less(yo ya) the spaceship to move is on a lower position
after checking these conditions we have to see if there is any spaceship in the middle of the trajectory
forall(sk ik) we check all spaceships and indexes
not(
and(
on(sk xab ik) spaceship is on the same x and some y
less(ik ya) check if it interferes with moved spaceship y
les(yb ik) check if it interferes with objective spaceship y
)if all these conditions are met then we cant satisfy the preconditions because its negated.
)
)
effect:
on(sa xab yo) we change y to y objective
not(on(sa xab ya)) we remove the previous position

go-down(sa spaceship, xab idx, ya idx, sb spaceship, yb idx, yo idx)

preconditions:

and(

on(sa xab ya)

on(sb xab yb)

one-less(yo yb)

less(ya yo)

forall(sk ik)

not(

and(

on(sk xab ik)

less(ya ik)

les(ik yb)

)

)

)

effect:

on(sa xab yo)

not(on(sa xab ya))

go-left(sa spaceship, xa idx, yab idx, sb spaceship, xb idx, yo idx)

preconditions:

and(

on(sa xa yab)

on(sb xb yab)

one-less(xb xo)

less(xo xa)

forall(sk ik)

not(

and(

on(sk ik yab)

less(ik xa)

les(xb ik)

)

)

)

effect:

on(sa xo yab)

not(on(sa xa yab))

go-right(sa spaceship, xa idx, yab idx, sb spaceship, xb idx, yo idx)
preconditions:
and(
on(sa xa yab)
on(sb xb yab)
one-less(xo xb)
less(xa xo)
forall(sk ik)
not(
and(
on(sk ik yab)
less(xa ik)
les(ik xb)
)
)
)
effect:
on(sa xo yab)
not(on(sa xa yab))

2 Explain the number of possible states

We have 25 positions and 6 spaceships that can be placed so we consider the following formula for combinations:

$$C(n,r) = \frac{n!}{(r!(n-r)!)} = ?$$

C(25, 6) = 177,100

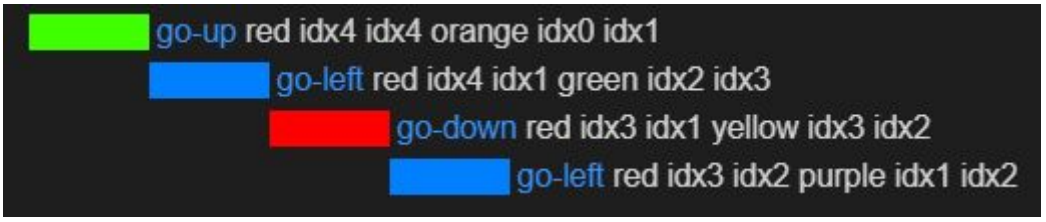
This is the complete number of states that could be defined with all the spaceships in the different positions, some of them probably wouldn't have a solution and the position of the red spaceship would be relevant but they could all exist as initial states.

3 PDDL Code for the planer

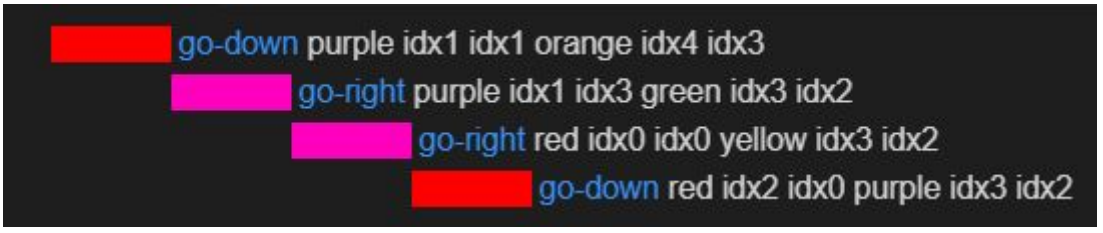
The domain is defined in **domain.pddl** and each of the problems is defined on **problem1.pddl** and **problem2.pddl**

The pddl solver is the one provided using the vs code pddl extension by default

Solution 1



Solution 2



Extra problem (problem39.pddl)

An extra problem has been defined with a few pictures that a classmate shared with us.

The solution provided by the planer is not exactly the same but it has been tested manually to be a valid one.