Bernat Martínez Gómez

World Definition

Objects

The world defined to solve our plainification problem consists of:

- n*n position objects
- one robot object
- one ghost object

Predicates

To represent the state of our domain the following predicates and numerical functions have been defined:

```
(:predicates (robot-on ?r - robot ?p - pos)
             (up ?p0 -pos ?p1 - pos)
             (left ?p0 -pos ?p1 - pos)
             (ghost-on ?g - ghost ?p - pos)
             (ghost-last ?g - ghost ?p - pos)
             (done ?r - robot)
)
(:functions
    (total-cost) - number
    (t-robot ?r - robot) - number
    (t-ghost ?g - ghost ?p - pos) - number
)
```

The board positions are defined using up(p0 p1) and left(p0 p1) to associate them in a top left to bottom right fashion.
To represent the robot state t-robot and robot-on are used, using both the robot position can be represented with an associated time dimension to it.

In order to define the ghost trajectory on the problem two predicates and one function have been defined: ghost-on(ghost position), ghost-last(ghost position) and t-ghost(ghost position). The ghost-on predicate is used on a similar fashion that the robot-on, the main difference can be found on the function t-ghost where each value is associated to both the ghost and a position object, by doing that a readable way to describe a path on the time dimension can be achieved. The ghost-last predicate has been included so if the ghost ever comes to a full stop the predicate can be used to describe it and there is no need to use more ghost-on predicates.

Finally, to have readable condition on when the problem is solved a done predicate has been defined in order to use it as goal. A total cost function has also been included in order to call for its minimization on the problem.

Bernat Martínez Gómez

Actions

Only six actions are needed on this planification problem, all focused on the robot

- Move right (robot pos0 pos1)
- Move left (robot pos0 pos1)
- Move down (robot pos0 pos1)
- Move up (robot pos0 pos1)
- Wait (robot pos0)
- Find (robot pos0 ghost)

All the move actions are defined in the same structure but changing some of the preconditions and effects, the basic concept is that the robot only can move to a cell next to the one sitting and the effect of doing so is removing the old robot-on predicate and adding the new one. On each action the total cost and the robot time dimension are increased by one. The wait action makes the robot stay on the same cell but increase the time value by one.

```
(:action move-up
    :parameters (?r - robot ?p0 - pos ?p1 - pos)
    :precondition (and (robot-on ?r ?p0) (up ?p1 ?p0))
    :effect (and (robot-on ?r ?p1) (not(robot-on ?r ?p0))
                 (increase (total-cost) 1)
                 (increase (t-robot ?r) 1))
)
(:action wait
    :parameters (?r - robot ?p0 - pos)
    :precondition (robot-on ?r ?p0)
    :effect (and (not(robot-on ?r ?p0))(robot-on ?r ?p0)
                 (increase (total-cost) 1)
                 (increase (t-robot ?r) 1))
)
```

The find action can only be done when the robot is in the same position that the ghost on that same timestamp, there is one extra condition for the case that the ghost last position was on said position and the time has already gone higher which means that it hasn't moved.

```
(:action find
    :parameters (?r - robot ?p0 - pos ?g - ghost)
    :precondition (or(and (robot-on ?r ?p0)(ghost-on ?g ?p0)
                          (=(t-robot ?r)(t-ghost ?g ?p0)))
                     (and (robot-on ?r ?p0)(ghost-last ?g ?p0)
                          (>=(t-robot ?r)(t-ghost ?g ?p0))))
    :effect (done ?r)
)
```

Problem definition example

Here a basic problem is explained as an example

```
(define (problem robot) (:domain room)
(:objects p0 p1 p2 p3 p4 p5 p6 p7 p8 p9 p10 p11 p12 p13 p14 p15 - pos
          r - robot
          g - ghost
)

(:init (up p0 p4)(up p1 p5)(up p2 p6)(up p3 p7)
       (up p4 p8)(up p5 p9)(up p6 p10)(up p7 p11)
       (up p8 p12)(up p9 p13)(up p10 p14)(up p11 p15)

       (left p0 p1)(left p1 p2)(left p2 p3)
       (left p4 p5)(left p5 p6)(left p6 p7)
       (left p8 p9)(left p9 p10)(left p10 p11)
       (left p12 p13)(left p13 p14)(left p14 p15)
       (=(total-cost) 0)
       (robot-on r p0)
       (=(t-robot r) 0)
       (ghost-on g p15)
       (=(t-ghost g p15) 0)
       (ghost-on g p11)
       (=(t-ghost g p11) 1)
       (ghost-on g p10)
       (=(t-ghost g p10) 2)
       (ghost-last g p10)
)
(:goal (done r))
(:metric minimize (total-cost)))
```

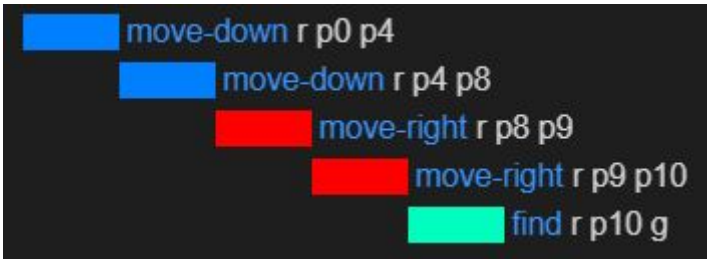
Initially the positions are defined and related to each other. After that, the total cost is set to 0 and the robot initial time and positions are defined.
Secondly, the ghost path is defined associating positions to it with their respective timestamps and the ghost-last predicate is used after the last position is written.
Finally, the goal is defined using the predicate that is generated when the find action succeeds and the metric minimize instruction is associated to the total cost value.

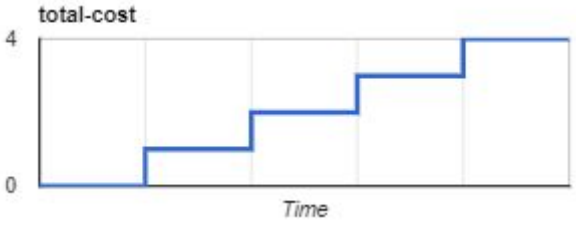
Bernat Martínez Gómez

Code and results

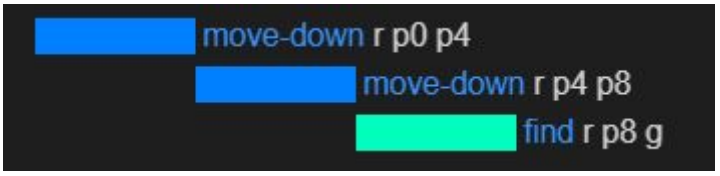
The domain is defined in domain.pddl and three examples of problems can be found. The 3rd one is using a 10 x 10 grid.

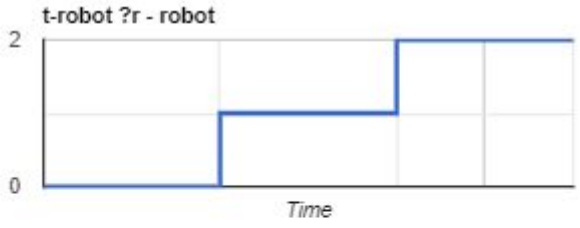
Problem 1 solution

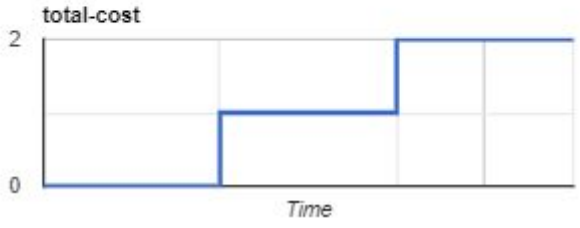




Problem 2 solution







Bernat Martínez Gómez

Problem 3 solution