Exercise 3

Moving Target

Problem Description

In this problem a point robot has to catch a moving target. Both the robot and the target live in an *N* x *N* 4-connected 2D grid-world, as depicted in Figure 1. This means they can only move up, down, left or right. Additionally, both the robot and the target can also decide to remain in the current position. However, neither the robot nor the target can move through the outer walls of the grid. Each cell in the grid-world is associated with the cost of visiting it. This cost is an integer that equals 1. Note that when the robot chooses to remain in the same cell, it will pay the cost of the visit again.

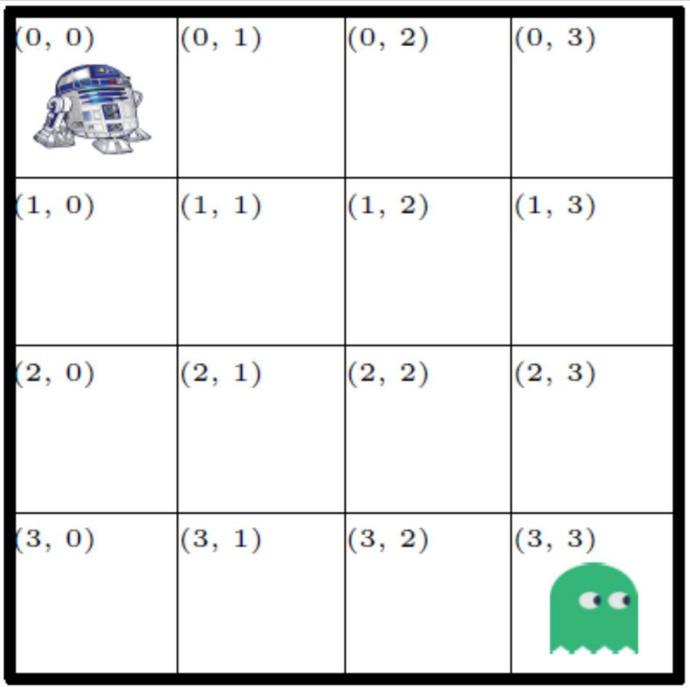


Figure 1: The robot and the target in the 2D grid-world.

The robot knows in advance the predicted trajectory of the moving target, as a sequence of positions in the grid (for example: (3, 3), (2, 3), (2, 2)). The first element of this trajectory is the initial position of the target at time step 0. Both the robot and the target move at the speed of one cell per time-step.

In this assignment, you will be writing the required PDDL files for a planner to help the robot catching the moving target. The task of the planners is to generate the path of the robot should follow in order to catch the target. You could suggest a planner in x and y domain, but if you can implement it with x, y and t (time-step) domain, there a bonus degree for that.

PDDL Representation

We will implement the planner in a x,y,t space. For doing so, we will define a series of predicates, grouped by the concept of the world we model:

- (end): This predicate becomes true through an action that checks ghost and player are in the same position. We will de_ne an action to set this predicate to true, and it is a very nice way to simplify the goal.
- (just less ?i - gridindex ?j - gridindex): These predicates allow to check grid ordering, necessary to check where can the robot move.
- (at ?e - entity ?i - gridindex ?j - gridindex): useful for tracking where each entity (be it the robot or the ghost) is at a time of the planning step.
- The two following predicates track whether the entities have already moved in the 't' the planner is currently in: (movedghost), (movedrobot),
- For tracking the ghost's sequence of movements, we define three predicates.

One is for ordering 'timesteps', which are identifiers for the sequence. The second one associates the position of the sequence with the index of the sequence (the timestep). The third one is for keeping track of which is the current element of the sequence. The problem could be easily simplified if we were sure that, given a sequence, there are no two positions that are the same. But if it can happen, this many predicates are necessary.

- (just before ?i - timestep ?j - timestep)
- (ghostpos ?t -timestep ?i - gridindex ?j - gridindex)
- (currentstep ?t - timestep)

We also define a cost metric and a time metric, which we keep updated through the actions.
As for the actions, we define 7 distinct actions:

- 1 is for 'advancing time', which allows us to reset the 'moved_entity' predicates whilst adding 1 to the time. This particular action has a prerequisite to have 'moved' the robot, which is to make it pay. Moving can be staying still, but that means paying the cost of the current position.
- 5 others are for moving the robot and making it pay the cost (this includes staying still, which still requires to pay the cost of the current position).

We also define an action for moving the ghost, which is a singular action since the ghost can only move from one position to the next in its predefined sequence. Finally, we define the aforementioned condition that requires both the robot and the ghost to be in the same position to set to true the (end) predicate.

Number of possible States

For the total number of possible states, we can have the robot be in every position of the board (N*N), and at any time up until the highest possible one (N*N*2) which is implicit to the problem definition. In reality, this is slightly less, though since the bot can only be in those positions it could reach, but the difference is insignificant. Furthermore, at any of those moments, the ghost can be in S positions, where S is the length of the sequence. In addition, we have one last, distinct state, which is the one that happens at the very end by setting the 'end' predicate. So the total number of predicates is:

- (N*N)*(2*N*N) from the robot (approximately).
- (S) From the ghost plus the current step predicate.
- +1 from the end predicate.

The total state configurations is approximately $(2 \times N^4 \times S) + 1$.
We finally wrap up by adding a metric optimisation target, which is the cost. The cost is added at every move action of the robot.

Domain and problems PDDL files

Domain PDDL file:

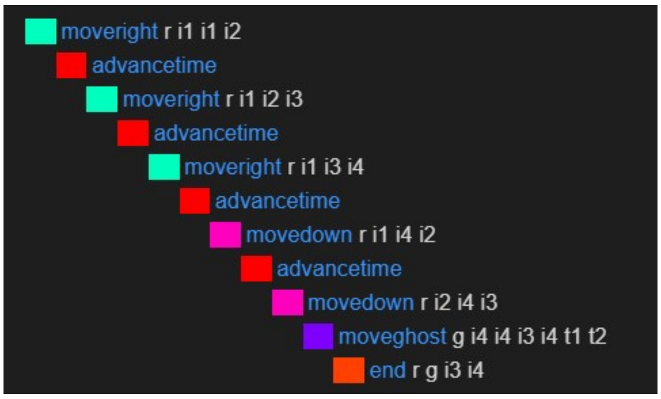
```
(define (domain ghostandrobot)
  (:requirements :strips :adl :typing :fluents)
  (:types
    gridindex - object
    entity - object
    robot ghost - entity
    timestep - object
  )
  ; Two gridindex identify a position in the board.
  ; Robot and ghost are the entities that move through positions
  ; timestep is an identificator for the sequence of movements of the ghost

  (:predicates
    (end); This predicate becomes true through an action that checks ghost and player are in the same position.
    (just_less ?i - gridindex ?j - gridindex) ; Example: just_less 1 2. just_less 1 4 is false.

    (at ?e - entity ?i - gridindex ?j - gridindex) ; Entity e is at coordinates i, j
    ; The two following predicates tell whether I can move the entities BEFORE advancing the time.
    (movedghost)
    (movedrobot)

    ; These are for keeping the ghost in its route
    (just_before ?i - timestep ?j - timestep) ; Example: just_before 1 2. just_before 1 4 is false. Good for time manipulation
    (ghostpos ?t - timestep ?i - gridindex ?j - gridindex); This, together with the previous predicate, tells us what the sequence of movements is.
    (currentstep ?t - timestep)
  )

  (:functions
    (time)
    (cost)
    (positioncost ?i - gridindex ?j - gridindex) ; Moving or staying in position (i, j)'s cost.
  )
(:action advancetime
    :parameters ()
    :precondition (movedrobot) ; Required to make it pay, at least via a movestay
    :effect (and
              (not (movedghost))
              (not (movedrobot))
              (increase (time) 1)
```

```
            )
  )

  (:action movestay
    :parameters (?r - robot ?i - gridindex ?j - gridindex)
    :precondition (and
              (not (movedrobot))
              (at ?r ?i ?j)
            )
    :effect (and
              (movedrobot)
              (increase (cost) (positioncost ?i ?j))
            )
  )


  (:action moveup
    :parameters (?r - robot ?i - gridindex ?j - gridindex ?i2 - gridindex
)
    :precondition (and
              (not (movedrobot))
              (at ?r ?i ?j)
              (just_less ?i2 ?i)
            )
    :effect (and
              (movedrobot)
              (not (at ?r ?i ?j))
              (at ?r ?i2 ?j)
              (increase (cost) (positioncost ?i2 ?j))
            )
  )
  (:action movedown
    :parameters (?r - robot ?i - gridindex ?j - gridindex ?i2 - gridindex
)
    :precondition (and
              (not (movedrobot))
              (at ?r ?i ?j)
              (just_less ?i ?i2)
            )
    :effect (and
              (movedrobot)
              (not (at ?r ?i ?j))
              (at ?r ?i2 ?j)
              (increase (cost) (positioncost ?i2 ?j))
            )
  )
  (:action moveleft
    :parameters (?r - robot ?i - gridindex ?j - gridindex ?j2 - gridindex
)
```

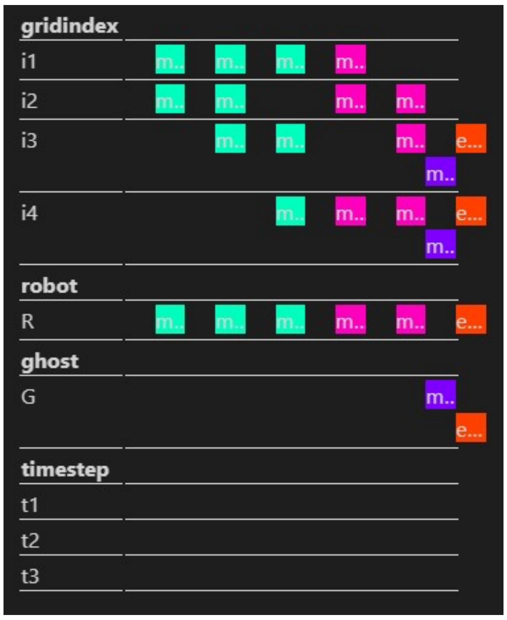
```
    :precondition (and
              (not (movedrobot))
              (at ?r ?i ?j)
              (just_less ?j2 ?j)
            )
    :effect (and
              (movedrobot)
              (not (at ?r ?i ?j))
              (at ?r ?i ?j2)
              (increase (cost) (positioncost ?i ?j2))
            )
  )
  (:action moveright
    :parameters (?r - robot ?i - gridindex ?j - gridindex ?j2 - gridindex
)
    :precondition (and
              (not (movedrobot))
              (at ?r ?i ?j)
              (just_less ?j ?j2)
            )
    :effect (and
              (movedrobot)
              (not (at ?r ?i ?j))
              (at ?r ?i ?j2)
              (increase (cost) (positioncost ?i ?j2))
            )
  )

  (:action moveghost
    :parameters (?g - ghost ?i - gridindex ?j - gridindex ?i2 - gridindex
 ?j2 - gridindex ?t -timestep ?t2 -timestep)
    :precondition (and
              (not (movedghost))
              (at ?g ?i ?j)
              (currentstep ?t)
              (just_before ?t ?t2)
              (ghostpos ?t2 ?i2 ?j2)
            )
    :effect (and
              (movedghost)
              (not (at ?g ?i ?j))
              (at ?g ?i2 ?j2)
              (not (currentstep ?t))
              (currentstep ?t2)
            )
  )
  (:action end
    :parameters (?r - robot ?g - ghost ?i - gridindex ?j - gridindex)
    :precondition (and
              (at ?r ?i ?j)
              (at ?g ?i ?j)
            )
    :effect (end)
  )
)
```

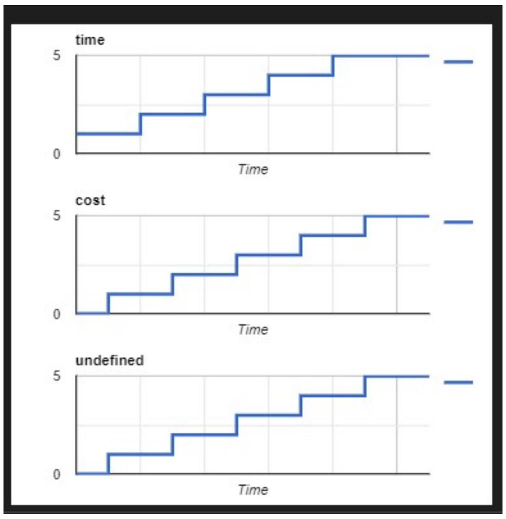
Problem PDDL file

```
; N=4
; all positions cost 1
; Ghost trajectory is: (3, 3), (2, 3), (2, 2)    ; 3,3 is lower-
right board position. In the representation, it is i4, i4.


(define (problem static1)
  (:domain ghostandrobot)
  (:objects
    i1 i2 i3 i4 - gridindex
    R - robot
    G - ghost
    t1 t2 t3 - timestep
  )
  (:init
    (just_less i1 i2)(just_less i2 i3)(just_less i3 i4)

    (just_before t1 t2)(just_before t2 t3)
    (currentstep t1)
    (ghostpos t2 i3 i4)(ghostpos t3 i3 i3)
    (at R i1 i1)(at G i4 i4)

    (= (time) 1)
    (= (cost) 0)
    (= (positioncost i1 i1) 1)(= (positioncost i1 i2) 1)(= (positioncost
i1 i3) 1)(= (positioncost i1 i4) 1)(= (positioncost i2 i1) 1)(= (position
cost i2 i2) 1)(= (positioncost i2 i3) 1)(= (positioncost i2 i4) 1)(= (pos
itioncost i3 i1) 1)(= (positioncost i3 i2) 1)(= (positioncost i3 i3) 1)(=
 (positioncost i3 i4) 1)(= (positioncost i4 i1) 1)(= (positioncost i4 i2)
 1)(= (positioncost i4 i3) 1)(= (positioncost i4 i4) 1)

  )
  (:goal
    (end)
  )
  (:metric
    minimize (cost)
  )
)
```



```
moveright r i1 i1 i2
  advancetime
    moveright r i1 i2 i3
      advancetime
        moveright r i1 i3 i4
          advancetime
            movedown r i1 i4 i2
              advancetime
                movedown r i2 i4 i3
                  moveghost g i4 i4 i3 i4 t1 t2
                    end r g i3 i4
```





Exercises

1. **Write a problem PDDL file for a 6x6 grid problem?**

2. **Evaluate the domain PDDL file and the two problem PDDL files of the two problems with existing planners like: FF planner (see the following link: https://www.cs.upc.edu/~jvazquez/teaching/iag/laboratorio/run_ff.html), or GraphPlan planner see the following link: https://www.cs.cmu.edu/~avrim/graphplan.html)**