INTELIGENCIA ARTIFICIAL

PRACTICA 2

Daniel, Garduño Hernandez
Mario Valdemaro, Garcia Roque

Pseudocodigos.

EJERCICIO 1

(defun f-goal-test-galaxy (state planets-destination))

- if state = null:
  NIL
- else:
  Se devuelve le resultado de la búsqueda de state en la lista planets -destination.

EJERCICIO 2

(defun f-h-galaxy (state sensors))

- if state = null or sensores = null:
  NIL
- else:
  Se devuelve la heurística que hay para el estado dado.

EJERCICIO 3

(defun navigate-white-hole (state white-holes))

- if state = null or white- holes = null:
  NIL
- else:
  Busca en la lista de white-holes si existe un entrada para state, si existe crea una accion en la que pone como origen state, como nodo destino el otro estado que aparezca en la entrada de la lista que hemos sacado y asignando un coste a la acción de ir de un estado a otro.

(defun navigate-worm-hole (state worm-holes))

- Si state = null or worm- holes = null:
  NIL
- else:
  Busca en la lista de worm-holes si existe un entrada para state, si existe crea una accion en la que pone como origen state, como nodo destino el otro estado que aparezca en la entrada de la lista que hemos sacado y asignando un coste a la acción de ir de un estado a otro.

EJERCICIO 4

(setf *A-star*)

Define una estrategia de busqueda a estrella, llamando a la funcion (node-f-<=)

(defun node-f-<= (node-1 node-2))

- If node-1 > node-2:
  NIL
- else:
  T

EJERCICIO 5

(setf *galaxy-M35*)

Define un problema de una galaxia asignando los *white-holes*,*worm-holes* y *sensors*

EJERCICIO 6

(defun expand-node (node problem))

- If node = null or problem = null:
  NIL
- else:
  Busca en problem el nodo que le hemos pasado:
  Si lo encuentra saca los elementos de las listas worm-holes, white-holes y sensors, con estos elementos se extraen los costes y las heurísticas y con ellos se asignan a una lista de estructuras nodes que se devolverá, donde los datos que se rellena son el padre (que es node), la acción (que consiste en llamar a la función del ejercicio 3), g que sera la suma del coste de desplazarse de un nodo a otro mas la “g” que arrastre nodo, “h” que sera la heurística que hemos sacado de sensores y “f” que es la suma de los datos anteriores. Este proceso se repetirá para cada elemento que hayamos extraído de worm-holes y white-holes.

Aclaraciones: se puede ver en el código que hay muchas comprobaciones en las sumas, ya que registramos un error en algunos casos en los que hacia una suma con NIL, lo que falla, por lo que decidimos poner comprobaciones de si el elemento que vamos a sumar es NIL o no.

Esto hace el código algo ilegible pero para facilitar la comprensión hay que pensar que cada argumento que se le pase a la suma sera de la forma:

(if(null x) 0 x)

Por otro lado a la hora de hacer la suma de “f” en vez de hacerlo de forma directa, sumar los valores de “g” y “h”, se utilizan los contenidos de las operaciones que de las operaciones para obtener “g” y “h” y se suman.

EJERCICIO 7

(defun insert-nodes-strategy (nodes lst-nodes strategy))

- If nodes, lst-nodes or strategy = null:
  NIL
- Else:
  Inserta de forma ordenada la lista de nodos nodes en la lista de lst-nodes mediante la estrategia strategy. La forma de hacerlo sera mediante una función (sort) que comprobara que dados dos nodos si la estrategia devuelve NIL o T y dependiendo del resultado los ordenadara de una forma o de otra.

EJERCICIO 8

(defun graph-search (problem strategy))

- Llama a la función graph-search-aux pasándole como argumentos “problem”, “strategy”, el nodo inicial y una lista vacía.

(defun graph-search-aux (problem strategy open-nodes closed-nodes)

- Si la lista open-nodes está vacía:
  terminar [no se han encontrado solución]
- Extraer el primer nodo de la lista open-nodes o si dicho nodo cumple el test objetivo:
  evaluar a la solución y terminar.
- En caso contrario:
- Si el nodo considerado no está en closed-nodes o es inferior al que está en closed-nodes de acuerdo con la estrategia strategy:
  expandir el nodo e insertar los nodos generados en la lista open-nodes de acuerdo con la estrategia strategy. incluir el nodo recién expandido al comienzo de la lista closed-nodes.
- Continuar la búsqueda eliminando el nodo considerado de la lista open-nodes.

EJERCICIO 9

(defun a-star-search (problem))

Llama a la función graph-search con la estrategia A estrella definida anteriormente.

EJERCICIO 10

(defun tree-path (node))

Llama a tree-path-aux con node y una lista vacia.

(defun tree-path-aux (node states))

- If node = null:
  Devuelve states.
- else:
  Inserta recursivamente el nombre de los ancestros y de node en states y devuelve el resultado.

EJERCICIO 11

(defun action-sequence (node))
Devuelve la llamada a la función action-sequence-aux que se llama con node y una lista vacia.

(defun action-sequence-aux (node actions))

- if node = null:
  Devuelve actions.
- if acción del nodo = null:
  Devuelve actions.
- else:
  Se llama recursivamente a si misma añadiendo en action todas las acciones de los nodos hasta el último ancestro.

EJERCICIO 12

(defun depth-first-node-compare-p (node-1 node-2))

- if node-1 or node-2 = null:
  Devuelve NIL.
- else:
  Compara las profundidades de los dos nodos
  - if node-1 > node-2
    T
  - else:
    NIL

(defun depth-first-node-compare-p (node-1 node-2))

- if node-1 or node-2 = null:
  Devuelve NIL.
- else:
  Compara las profundidades de los dos nodos
  - if node-1 > node-2
    NIL
  - else:
    T