Sandu Mihai Alexandru 0321308

Domanda n° 3

1) Descrizione dell'es in termini di PEAS:

P: Performance -> Il robot deve andare in magazzino, prendere i materiali necessari, successivamente andare nella stanza indicata per poter costruire il mobile

E: Enviroment -> Abbiamo uno spazio interno strutturato con le varie stanze e i corridoi che le collegano, la presenza di ostacoli come muri, questi ostacoli sono fissi

A: Attuators -> Motori per lo spostamento del robot e braccia per la costruzione del mobile

S: Sensors -> Sensori per evitare gli ostacoli come una telecamera

2) Rappresentazione della mappa in una struttura dati adeguata

Possiamo rappresentare la nostra mappa come un grafo dove:

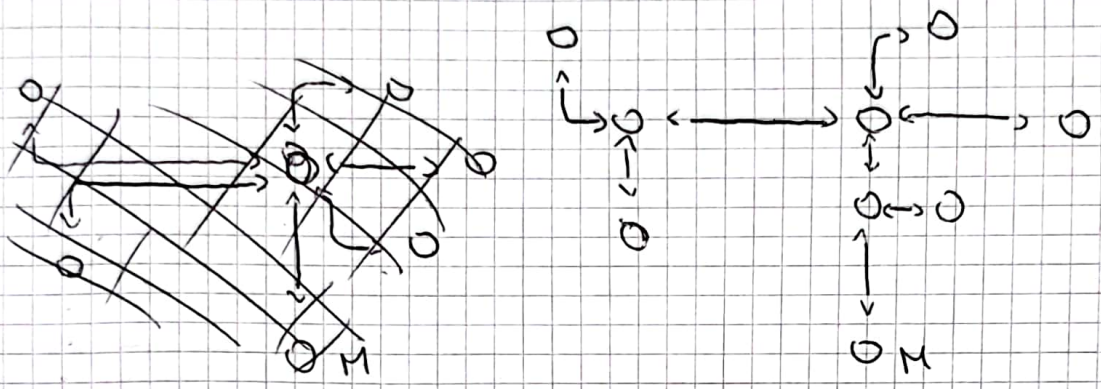
- i nodi rappresentano le posizioni chiave come stanze, incroci e porta del magazzino
- gli archi rappresentano tutti i percorsi possibili tra i nodi.

E' importante nella modellazione dell'ambiente non rendere il grafo connesso poiché il robot costruito un mobile la sua unica direzione sarà il magazzino e non altre stanze. Inoltre poiché ci troviamo di fronte a un ambiente completamente osservabile e statico, cioé gli ostacoli sono fissi possiamo modellare gli archi e i nodi in modo che il robot non incontri i vari ostacoli.

L'ambiente oltre a essere completamente osservabile, e statico é anche: deterministico poiché dato uno stato e un azione ci corrisponde sempre lo stesso stato,
discreto, poiché in presenza di un numero di stati finiti;
e singolo agente, poiché il nostro unico agente é il robot.
L'ambiente inoltre é episodico poiché le azioni effettuate

per andare a costruire un ~~robot~~ mobile non influenzeranno le azioni che il robot userà per costruire il successivo mobile.

Il grafo sarà così rappresentato



Il nodo M rappresenta il magazzino mentre il nodo centrale abbiamo un incrocio gli altri nodi rappresentano o altri nodi oppure altri incroci.

Il peso dei singoli archi equivale alla distanza che il robot percorre

3) Nel nostro stato iniziale il robot sarà in una posizione del grafo in attesa di ordini per andare a prendere i materiali richiesti così da costruire il mobile, una volta presi i materiali quindi arrivato in magazzino partirà nella stanza richiesta

Nel nostro stato generale abbiamo il robot all'interno della mappa con una lista di materiali raccolti o ancora da raccogliere e una lista di operazioni di assemblaggio completate.

Nello stato finale il robot ha costruito il mobile nella stanza. corretta

4) L'algoritmo per la navigazione che andremo a utilizzare sarà l'algoritmo A*. Questo è un algoritmo ottimale e completo per grafi con pesi positivi e con A* va utilizzata l'euristica h(n) della distanza euclidea. Viene scelta quest'euristica poiché non è detto che ~~in altri~~ possiamo utilizzare altre euristiche come la distanza di Manahthan poiché non abbiamo una griglia o una matrice e neanche altre euristiche sarebbero compatibili.

5) Le azioni che il robot potrà intraprendere sono 6 azioni di spostamento: N, S, E, W

Ulteriori azioni sono per la costruzione del mobile: BUILD

e per la raccolta dei materiali: FETCH

Ovviamente l'azione di BUILD verrà fatta soltanto se ci troviamo nella giusta stanza mentre l'azione di FETCH verrà fatta soltanto se ci troviamo nel magazzino. Il costo delle azioni di spostamento variano in base all'arco percorso mentre il costo delle altre due azioni sarà pari a 0.

Definiamo ora:

NA: come posizione del robot M= magazzino

P: come la nostra stanza goal

[MAT]: lista dei materiali raccolti

DIST: distanza che il robot ha percorso

Il nostro stato iniziale sarà [NA=?, P=?, [MAT]=empty, DIST=0]

6) Scriviamo ora il codice del ciclo di vita dell'ambiente

```
function RUN-EVAL-ENVI(state, UPDATE-FN, agent, PERFORMANCE-FN)
        returns DIST
local variables DIST % inizialmente a 0
repeat
  (NA, P, MAT) <- GET-PERCEPT(agent, state)
  Action <- ROBI-PROGRAM((NA, P, MAT))
  state <- UPDATE-FN(Action, state)
  DIST <- PERFORMANCE-FN(DIST, state)
until TERMINATION(state)
returns DIST
```

Ora scriviamo il ciclo di vita dell'agente ROBY

function ROBYPROGRAM(NA, P, MAT) returns action

persistent:

plan, seq di azioni inizialmente vuota

goal, un goal inizialmente nullo

problem, formulazione del problema

state, stato del mondo corrente

if plan is EMPTY? then

$NG^* \leftarrow$ FORMULATE-GOAL(NA, P, [MAT])

problem $\leftarrow$ FORMULATE-PROBLEM((NA, P, [MAT]), NG^*)

plan $\leftarrow$ SEARCH(problem) % A^* con euristica dist euclidea h()

if plan = failure then return null action

if NA = NG^* e NG^* = M then action = FETCH

else if NA = P e NG^* = P e [MAT] ≠ EMPTY then action = BUILD

else

action $\leftarrow$ FIRST(plan)

plan $\leftarrow$ REST(plan)

return action

Come scritto nel problema della search andremmo a utilizzare l'algoritmo A^*. Quello che farà l'agente sarà quindi prendere i materiali dal magazzino M, una volta presi gli sarà dato l'ordine di andare a costruire il mobile nella stanza P. Tramite la search dell'algoritmo troverà il percorso migliore per la stanza P e una volta arrivato lì entrerà nel secondo if poiché la condizione sarà soddisfatta e utilizzerà l'azione BUILD per costruirlo. Successivamente a nuovo ordine il robot avrà come nodo goal il magazzino dove con il search troverà il percorso

migliore e una volta arrivato ad M tramite l'azione FETCH riempie la lista di materiali una volta riempita la lista troverà il nodo goal che corrisponde alla stanza alla quale deve arrivare e da lì ricomincia finché non avrà costruito tutti i mobili.

Definiamo ora lo pseudocodice di A*:

```
function A*(problem) returns solution or failure
  node <- nodo con stato = problem.INITIALSTATE
  frontiera <- coda di priorità ordinata da f(n) con all'interno inizialmente node
  esplorati <- insieme vuoto
loop do
  if frontiera is EMPTY? then return failure
  node <- pop(frontiera)
  if problem.GOALTEST = node.STATE then return SOLUTION(node)
  aggiungi node.STATE agli esplorati
  for each azione in problem.ACTIONS(node.STATE) do
     child <- CHILDNODE(problem, node, action)
     if child.STATE NON è in esplorati o frontiera then
       frontiera <- INSERT(frontiera, child)
     else if child.STATE is in frontiera con f(n) più grande then
      scambia quel nodo in frontiera con il child
```

La funzione f(n) menzionata nell'algoritmo A* è: $f(n) = g(n) + h(n)$ dove h(n) è la nostra euristica di distanza euclidea