Corso di laurea magistrale in
INFORMATICA

Corso di Intelligenza Artificiale e Laboratorio

Docenti: R. Micalizio, G. Pozzato, G. Torta

a.a. 2016/17

Progetto: Rescue 2017

OBIETTIVO

Il progetto in questione ha lo scopo di sviluppare un agente intelligente che sappia combinare aspetti deliberativi con aspetti reattivi per svolgere un compito complesso in un ambiente solo parzialmente osservabile e in presenza di altri agenti. Nel presente progetto ci si occupa solo della parte decisionale/cognitiva mentre tutta la parte sensoristica e attuativa è simulata (in modo molto approssimato) via software.

IL DOMINIO

Il dominio di riferimento (estremamente semplificato) prende lo spunto da problematiche di protezione civile in particolare l'esplorazione di un edificio per la localizzazione di persone ferite in seguito ad eventi naturali distruttivi (esempio terremoti).

Il compito da svolgere è un compito di ricognizione e si suppone possa essere svolto da un agente robotico mobile dotato di un'opportuna suite di sensori ed in grado di muoversi nell'ambiente da esplorare. Inoltre nella attività di rescue sono impegnati agenti umani che hanno compiti diversi ma che condividono con l'agente robotico lo stesso ambiente.

Il mondo in cui si deve muovere l'agente robotico è un edificio che per ragioni di semplicità viene assunto essere ad un solo piano e di forma rettangolare.
L'edificio può essere schematizzato come una griglia in cui ciascuna cella può contenere una parete *(wall*), una connessione interno/esterno *(gate*), essere ricoperta di macerie (*debris*), oppure essere vuota (*empty*), od ancora essere occupata da una persona (*person*). L'edificio è completamente circondato da celle di tipo *outdoor* che indicano la parte (immediatamente) esterna all'edificio.
Si suppone che l'edificio lesionato nel momento in cui l'agente robotico inizia la sua missione sia contornato completamente da celle di tipo *wall*, *gate*, *debris* e che ci sia almeno un *gate* (quello in cui si trova inizialmente localizzato il robot). Come detto l'edificio è circondato da celle *outdoor* che sono già state liberate da eventuali macerie.

In seguito alla calamità naturale è possibile che una o più persone siano rimaste sepolte sotto alle macerie. Si assume che solo in una cella di tipo *debris* si possa trovare una persona intrappolata e che in una cella coperta di macerie ci possa essere al massimo una sola persona.

IL COMPITO DELL'AGENTE ROBOTICO

Lo scopo dell'agente robotico è quello di individuare le celle in cui sono rimaste intrappolate le persone nel più breve tempo possibile.
Il compito è reso complesso dal fatto che le persone intrappolate non sono immediatamente visibili (sono coperte da macerie), ma emettono solo dei deboli suoni che il robot può catturare (con numerose limitazioni - vedi capitolo percezioni).
In particolare il robot una volta individuata una persona intrappolata deve informare (tramite apposita azione – vedi sotto) il Comando Operativo delle Attività di Soccorso (COAS). Inoltre nell'ambiente sono presenti dei soccorritori umani che si muovono per compiere compiti diversi (il vero e proprio rescue delle persone intrappolate). E' compito del robot informare il COAS anche di quali sono le celle di tipo *debris* in cui è possibile escludere la presenza di persone intrappolate (diremo che la cella è *checked*) . Infine, è bene che il robot informi il COAS di quali celle sono *clear* cioè quali celle possono essere percorse senza particolari precauzioni dai soccorritori umani essendo sgombre da macerie.

Al robot è assegnato un tempo massimo entro cui la missione di esplorazione e rescue deve terminare. Il robot ovviamente dovrà fare del suo meglio in questo lasso di tempo. Per rendere valutabile l'attività del robot è previsto un meccanismo di penalità (vedi apposito capitolo) che definisce in qualche modo l'utilità del comportamento del robot.

LE FUNZIONALITA' DELL'AGENTE ROBOTICO

Come accennato sopra, l'agente robotico è un robot di servizio dotato di opportune funzionalità tra cui quelle locomotorie (mobilità), sensoriali (vedi sotto) e di manipolazione (capacità di movimentare le macerie).

Il robot può muoversi in avanti in una delle quattro direzioni (north, south, west, east) e può cambiare direzione con una apposita manovra ruotando di 90 gradi a destra o a sinistra. Il robot non può andare ad occupare una cella che sia occupata da qualche altra entità.
Si noti che:

- Il robot ha spazio sufficiente per caricare a bordo i detriti relativi ad una sola cella. Ovviamente prima di poterne caricare degli altri deve svuotare il carico
- Quando è carico, il robot impiega più tempo e consuma più power per muoversi rispetto a quando è vuoto

Ad ogni istante di tempo il robot può fare una sola delle azioni sotto elencate:

Forward
Con questa azione il robot avanza di una posizione (cella) nella direzione in cui è attualmente orientato. Ovviamente l'azione di muoversi ha successo se la cella in cui il robot deve andare è di tipo *empty* o *gate*. Se si cerca di fare una azione di *forward* non permessa, l'azione fallisce nel senso che il robot rimane posizionato dove è, ma subisce un qualche danno (più sotto si misurano le prestazioni del robot in termini di penalty). Inoltre riceve una percezione specifica (*bump*, si veda sotto).
Questa azione richiede 1 unità di tempo per essere eseguita se il robot è scarico, 2 unità di tempo se è carico, 10 unità di tempo se l'azione porta al fallimento.

Turnright
Con questa azione il robot svolta su se stesso di 90 gradi a destra rispetto alla sua direzione corrente. Si tenga presente che le direzioni sono 4 (in particolare: north, south, east, west). Il robot non cambia cella con questa operazione. Questa azione non fallisce mai.
Questa azione richiede 2 unità di tempo per essere eseguita se il robot è scarico e 3 se il robot è carico.

Turnleft
Con questa azione il robot svolta su se stesso di 90 gradi a sinistra rispetto alla sua direzione corrente. Le condizioni di applicazione sono esattamente uguali a quelle di Turnright.
Questa azione richiede 2 unità di tempo per essere eseguita se il robot è scarico e 3 se il robot è carico.

Load_Debris (r,c)
Con questa azione il robot può caricare a bordo le macerie contenute in una cella di tipo *debris* con coordinate (r,c). Poiché il piano di carico del robot è limitato può caricare al massimo le macerie di una sola cella. Inoltre per poter eseguire l'azione, il robot deve essere posizionato in una cella adiacente alla cella di tipo *debris* (e nella cella non deve essere intrappolata una persona). Se l'azione è eseguibile, lo stato dell'ambiente cambia perché la cella (r,c) diventa *empty/gate*. L'azione può fallire perché il robot cerca di fare una operazione di *load_debris* in una cella non

adiacente, oppure il robot è già carico, oppure la cella da cui si vuole caricare non contiene *debris*. In tutti questi casi il robot può continuare a funzionare ma riceve severe penalità.
Esiste però un fallimento particolare di tipo catastrofico. Questo fallimento avviene quando il robot cerca di caricare *debris* sotto cui è imprigionata una persona. In questo caso l'esecuzione si interrompe e il robot non può più continuare la sua missione.
La durata di una azione di *load_debris* è di 20 unità di tempo.
Per le percezioni che l'ambiente emette in seguito all'azione di *load_debris* si veda sotto.

Unload_Debris(r,c)
Con questa azione il robot può scaricare le macerie che ha a bordo in una cella di tipo *empty* collocata in posizione (r,c). Inoltre per poter eseguire l'azione il robot deve essere in un cella accanto alla cella in cui decide di scaricare. Se l'azione è eseguibile, lo stato dell'ambiente cambia perché la cella che era *empty* diventa di tipo *debris*.
Il robot può anche scaricare in una cella di tipo *outdoor*. In questo caso la cella non cambia di stato visto che all'esterno ci sono altri mezzi per sgomberare quanto scaricato dal robot.
Se la cella in cui deve scaricare non è del tipo previsto oppure il robot cerca di scaricare in una cella non adiacente alla sua posizione, l'azione fallisce.
Si noti che è fisicamente possibile per il robot scaricare le macerie in una cella che è già stata dichiarata *clear* dal robot stesso. Tuttavia questa azione viene scoraggiata con l'attribuzione di penalità.
L'azione di scarico richiede 16 istanti di tempo se viene effettuata in una cella empty/gate e 10 se viene effettuata in una cella di tipo outdoor.
Per le percezioni che l'ambiente emette in seguito all'azione di *unload_debris* si veda sotto.

Drill(r,c)
Con questa azione il robot può praticare un foro per infilare una sonda nella cella coperta di macerie che si trova nella riga *r* e nella colonna *c* (il robot deve trovarsi in una cella attigua). Il fine ultimo di questa azione di manipolazione è eseguire una fase di sensing: l'effetto dell'azione è quella di poter acquisire informazioni precise sulla presenza o meno di una persona nella cella in cui si è scavato (perché la sonda permette di osservare e percepire l'eventuale persona intrappolata). L'operazione di *drill* è un'operazione complessa che prende 60 unità di tempo.
L'azione è fisicamente possibile anche dove non ci sono *debris*, ma l'agente riceve molte penalità perché l'azione oltre ad essere inutile rischia di danneggiare gli apparati robotici che sono responsabili dell'azione di drill.
Per le percezioni che l'ambiente emette in seguito all'azione di *drill* si veda sotto.

Inform(r,c,status)
Con questa azione il robot informa il COAS dello stato della cella di coordinate (r,c). L'informazione più rilevante che viene trasmessa è quella relativa alla presenza di una persona intrappolata sotto le macerie nella cella di posizione (r,c); in questo caso il valore di status è impostato a

discover. Informazioni rilevanti sono anche quelle relative ad una cella di tipo *debris* che risulta verificata (valore di status a *checked*) quando è stata esclusa la presenza di una persona intrappolata sotto le macerie. Infine il robot può informare che una cella è *clear* nel senso che è vuota (o è stata liberata), per cui i soccorritori si possono muovere liberamente in essa.
L'azione di *inform* non fallisce mai, ma ovviamente il suo impatto è ben diverso nel caso in cui l'informazione inviata sia corretta rispetto al caso in cui l'informazione inviata sia inesatta. Inoltre la stessa informazione inviata più volte è poco utile. Questi diversi casi saranno modellati con il meccanismo delle penalità (vedi oltre).
L'azione di *inform* viene eseguita in un solo istante di tempo.

Done
Con questa azione il robot informa il COAS che a suo giudizio il compito è terminato e non può fare più nulla di utile nel tempo a disposizione che è rimasto. L'azione di *done* non fallisce mai, ma dovrebbe essere eseguita solo quando il robot si trova in una cella di tipo *gate*. Anche in questo caso le penalità avranno un ruolo rilevante. Ovviamente dopo il messaggio di *done*, il robot cessa di operare.
L'azione di *done* viene eseguita in un solo istante di tempo.

Wait
Con questa pseudo azione l'agente robotico decide di attendere 5 unità di tempo senza fare nulla. Ovviamente questa azione non fallisce mai.

CONOSCENZA A PRIORI SULL'AMBIENTE

L'agente robotico ha una conoscenza a priori sull'ambiente in cui si trova ad operare molto limitata; in particolare conosce solo:

- la mappa dell'ambiente dall'esterno: conosce cioè solo le celle outdoor e solo le celle che sono a contatto con outdoor (che possono essere solo di tipo *wall*, *debris* e *gate*). Conosce pertanto le dimensioni dell'edificio e per tutte le celle interne non conosce né il tipo né se ci sono persone intrappolate.
- Conosce la sua posizione iniziale (una cella di *gate*) ed inoltre sa di essere scarico
- Sa quanto tempo avrà a disposizione per compiere missione (*maxduration*)
- Sa che l'ambiente non si modifica per dinamica interna. In altre parole non succedono crolli durante la sua fase di esplorazione per cui se una cella all'istante *t* è di tipo *wall* tale rimane nel seguito. Il mondo cambia solo per le azioni del robot stesso che può ripulire dalle macerie una cella.

Come risulta evidente da quanto detto sopra l'agente robotico opera in un ambiente complesso (per quanto drasticamente semplificato) in cui vi sono anche agenti umani (i soccorritori). Si assume che gli agenti umani abbiano un atteggiamento cooperativo con l'agente robotico, per cui quando sono in movimento e si vengono a trovare in una cella adiacente a quella del robot non vanno intenzionalmente ad urtare il robot, ma

stanno fermi fino a quando il robot non si allontana (ma è compito del robot non intralciare i soccorritori, si veda parte delle penalità) e possono continuare nel cammino da loro previsto.

Infine vale la pena menzionare quello che l'agente robotico **non sa a priori**: non sa quante persone siano rimaste intrappolate (potenzialmente anche nessuna) e a maggior ragione non sa dove siano intrappolate. E' proprio il compito dell'agente scoprire quante persone (e dove) sono rimaste intrappolate e informare di questo il comando dei soccorritori (COAS).

LE PERCEZIONI

Come nello schema classico, ad ogni istante l'agente riceve percezioni dal mondo esterno (nel nostro caso dall'ambiente che modella l'edificio colpito da calamità) tramite i suoi sensori. In questo dominio i sensori sono svariati e nel seguito verranno esaminati uno ad uno.

Percezione Visiva

Si suppone che l'agente abbia un sistema di visione (che non è soggetto a guasti) omnidirezionale per cui il robot riceve informazioni attraverso il sensore visivo sul contenuto delle otto celle contigue a quella in cui si trova.
In modo più preciso, se il robot si trova nella cella di coordinate (r,c) e la sua direzione è *north* riesce a vedere cosa contengono le celle (r+1,c-1), (r+1,c) (r+1,c+1) (r,c-1), (r,c), (r,c+1) (r-1,c-1), (r-1,c) (r-1,c+1). Analoghe considerazioni valgono per le altre direzioni.
Le percezione visiva perviene alla parte decisionale del robot già classificata mediante uso di opportuni algoritmi di classificazione ed interpretazione. Si suppone che gli algoritmi di interpretazione siano tanto sofisticati e precisi da fornire per ogni cella l'informazione in termini di:

- *empty*
- *debris*
- *wall*
- *gate*
- *person*
- *outdoor*

La percezione relativa alla cella di coordinate (r,c) conterrà *robot* poiché in quella cella è situato al momento l'agente robotico.
Si noti che il sistema di visione non è in grado di vedere una persona intrappolata sotto le macerie (l'interpretazione della cella rimane *debris*) per cui la label *person* fa riferimento ad un soccorritore che si muove nell'ambiente.
Per semplicità si assume che la percezione visiva venga fornita al termine di qualunque azione eseguita dal robot (anche la *inform* e la *wait* , ma non la *done* che determina la conclusione delle attività).

Percezioni acustiche
Si suppone che il robot sia inoltre fornito di un sensore acustico che è in grado di percepire rumori che provengono da una persona che sia intrappolata in una delle 4 celle adiacenti alla posizione in cui si trova il robot.
Si noti che questa informazione non è direzionale per cui se il robot si trova nella cella di coordinate (r,c) il sensore acustico è in grado di percepire rumori se almeno una persona si trova sotto le macerie nelle celle di coordinate (r+1,c), (r-1,c), (r,c), (r,c+1), (r,c-1), cioè nelle celle contigue a quella in cui si trova il robot. Si noti che la percezione non è graduata, per cui la percezione è la stessa sia nel caso ci sia una sola persona nelle 4 celle adiacenti che due o tre.

L'agente robotico ha la possibilità di acquisire deliberatamente informazioni aggiuntive con una apposita azione di *sensing*: infatti, l'azione di *drill* ha l'effetto di poter introdurre una sonda e tale sensore è in grado di dare informazione precisa sulla presenza o meno di una persona nella cella in cui si è scavato (*yes* o *no*). Si noti che se si compie una azione di *drill* in una cella in cui non ci sono *debris*, la percezione è *fail* (e ci sono penalità).

Percezioni di ostacolo
Infine se il robot tenta di fare una azione di *forward* in una cella che non sia *empty* o *gate*, riceve una percezione di *bump*.

Percezioni di carico
Il robot ha anche un sensore di carico per cui (solo dopo azioni di *load* o di *unload*) riceve una percezione codificata come *yes* oppure *no* a seconda che il robot sia carico o scarico.

CRITERI DI UTILITA'

Per permettere una valutazione comparativa tra diverse strategie decisionali adottate dall'agente, nel sistema è previsto un meccanismo di penalità che fornisce una valutazione dell'efficienza (o inefficienza) dell'agente nel risolvere un dato problema.
Poiché il tempo richiesto per localizzare una persona imprigionata tra le macerie è di fondamentale importanza, le penalità sono particolarmente elevate in presenza di persone sotto le macerie.
In particolare:

- 50 punti di penalità per ogni istante di tempo e per ogni persona intrappolata sotto le macerie. L'assegnazione di penalità per una persona sotto le macerie si interrompe nell'istante in cui l'agente esegue una azione di *inform* relativa alla persona in questione.

Altri aspetti sono inoltre tenuti in considerazione:

- 4 punti di penalità per ogni istante di tempo per ogni cella di tipo *debris* che non contenga una persona intrappolata. L'assegnazione di penalità cessa nel momento stesso che viene eseguita azione di *inform* che è *checked*.
- 1 punto di penalità per ogni istante di tempo per cui una cella è *empty* fino a quando questa non viene dichiarata *clear*.

- 5 punti di penalità per ogni istante di tempo in cui una persona (soccorritore) deve stare ferma per lasciare transitare il robot

Mentre almeno in parte queste penalità sono inevitabili nel senso che l'agente robotico non può fare più attività in parallelo, vi sono altre penalità che l'agente può facilmente evitare con un comportamento "smart".

Nel caso in cui l'*inform* sulla presenza di una persona intrappolata sia fallace (non c'è persona nella cella indicata) questa viene considerata un falso allarme che viene punito con una penalità di 20.000.000 punti. Se una cella contenente *debris* viene data come *checked* quando invece c'è una persona intrappolata riceve una penalità di 50.000.000 punti; se viene data come *checked* una cella che non contiene *debris* , la penalità è di 5.000.000 punti.
Nel caso in cui una cella sia dichiarata erroneamente *clear* si riceve una penalità di 200.000 punti.
Nel caso in cui l' *inform* sia corretta, ma l'informazione sia già stata comunicata in precedenza, si ha una penalità di 25.000 punti.

Per quanto riguarda le penalità relative a movimenti sbagliati, azioni di *forward* che portano ad urti con *wall*, *debris* o *outdoor* (percezione di *bump*) vengono punite con una penalità di 1.000.000 punti (azioni sbagliate di questo tipo possono danneggiare il robot). Se invece l'azione porta ad urtare un soccorritore (*person*) le penalità ammontano a 10.000.000 punti.

Le azioni di *drill* sono delicate perché comportano l'uso di strumenti sofisticati e fragili. Per questa ragione un'operazione di *drill* che fallisce (per una delle ragioni esposte nella descrizione dell'azione stessa) viene punita con una penalità di 1.000.000 punti. Si noti che una azione di *drill* effettuata in una cella che contiene *debris* ma senza una persona intrappolata non è un fallimento, perché eseguire una azione di *drill* è uno dei meccanismi per accertarsi della presenza di una persona. Si noti che in alcune situazioni tramite opportuni meccanismi di ragionamento si può inferire con certezza la presenza di una persona in una cella senza dover fare una *drill* (quindi risparmiando azioni e di conseguenza ricevendo minori penalità).

Esistono anche penalità per le operazioni di *load_debris* e *unload_debris* che falliscono per una delle ragioni riportate nella descrizione delle azioni. In questo caso viene applicata una penalità di 1.000.000 punti. Si noti che il tentativo di ripulire una cella quando questa è occupata da una persona (soccorritore) viene punita con 10.000.000 punti, così come il tentativo di scaricare in una cella occupata da un soccorritore.
Si noti che scaricare il carico in una cella *empty* che è stata dichiarata *clear* comporta una penalità di 100.000 punti.

Il robot deve dichiarare di aver completato il compito assegnatogli prima della scadenza del tempo concessogli (*maxduration*) tramite il messaggio di *done*. Come detto sopra, l'azione di *done* indica la cessazione della attività del robot (e quindi lo stop ai conteggi delle penalità dovute al trascorrere del tempo). Tuttavia se il messaggio di *done* è inviato dall'agente quando non tutte le persone sotto le macerie sono state localizzate (e la loro posizione è stata comunicata al COAS tramite azioni di *inform*), per ogni persona non localizzata è assegnata una penalità di 2.000.000 punti.

Inoltre, se al momento del done ci sono delle celle contenenti *debris* che non sono etichettate come *checked* la penalità è di 200.000 punti.
Infine se l'agente robotico decide di inviare il messaggio di *done* e non si trova in un *gate* riceve una penalità di 100.000 punti.

Se invece termina il tempo a disposizione (dato da *maxduration*) senza che il robot abbia comunicato *done*, l'attività si interrompe ma il robot riceve una penalità aggiuntiva di 10.000.000 di punti.

Si noti che una comparazione sensata tra strategie diverse basata sul meccanismo delle penalità è possibile solo a parità di problema da risolvere, dove per problema non si intende solo la dimensione dell'edificio, ma anche il numero e la dislocazione delle celle di *debris*, il numero di persone intrappolate in tali celle, la posizione iniziale del robot. Quindi il punteggio non deve essere preso in assoluto, ma relativamente alla caratteristiche del dominio. Ad esempio, un ambiente con molti ostacoli è più difficile da navigare di un ambiente delle stesse dimensioni con pochi ostacoli, inoltre i movimenti del robot possono essere rallentati o impediti se i soccorritori sono numerosi e si muovono molto.
Infine un parametro essenziale da considerare è il tempo concesso per risolvere il problema (*maxduration*).

Queste considerazioni vanno tenute in conto in fase di testing del sistema (si veda apposito capitolo).

L'IMPLEMENTAZIONE

Al fine di semplificare lo sviluppo concettuale e implementativo di Rescue 2017, si mette a disposizione degli studenti un ambiente di simulazione in CLIPS per lo sviluppo dell'intero sistema (agente + simulatore dell'ambiente). Questo ambiente prevede tre moduli base:

- *MAIN* è usato esclusivamente per la comunicazione tra gli altri due moduli
- *ENV* che modella l'ambiente in cui opera il robot (compresi i movimenti delle persone)
- *AGENT* che modella il supervisore del robot

Più alcuni moduli di supporto di *AGENT*:

- *PERCEPTION*
- *REASONING*
- *ACTION*

Si noti che il sistema è molto complesso in quanto non solo deve gestire il ciclo tipico di un agente che riceve percezioni dall'ambiente, decide cosa fare e manda in esecuzione opportune azioni tramite gli attuatori, ma deve tenere in conto che l'ambiente è condiviso da più agenti ciascuno con una visione parziale e limitata dello stato del mondo.

Anche se qualche maggiore dettaglio verrà fornito quando si parla di ENV, vale la pena notare che sullo stato del mondo esistono almeno due visioni:

- Visione del simulatore (codificato nell'ENV) che ovviamente deve contenere lo stato "VERO" del mondo. Da questo punto di vista ENV è omnisciente perché sa fin dall'inizio dove sono le persone intrappolate. Questa visione è dinamica perché l'agente robotico svolge azioni i cui effetti vanno a cambiare lo stato del mondo.
- Visione dell'agente (che verrà codificata in AGENT) che è molto parziale all'inizio (conoscenza scarsa a-priori sul mondo) e che cresce se l'agente è in grado di sfruttare le percezioni e di ragionare sugli effetti delle azioni che compie.

Una terza visione è quella del COAS che condivide all'inizio la stessa visione molto parziale dell'agente robotico e che evolve nel tempo sulla base delle informazioni trasmesse dal robot stesso (solo per semplicità di implementazione si assume che i soccorritori non aggiungano informazioni) .
Questa terza visione non è gestita esplicitamente nel codice di ENV, anche se nelle attribuzioni delle penalità si tiene conto di cosa è stato detto al COAS in precedenza.

E' compito ancora dell'ENV quello di aggiornare due parametri importanti quali *time* e *step*. *Time* è l'informazione del tempo che trascorre relativo all'inizio della attività ed è misurato in unità di tempo (si può assumere che sia dell'ordine dei secondi). *Step* invece è un contatore (inizializzato a zero) che tiene traccia del numero di cicli percezione/decisione/attuazione finora fatti dall'agente robotico. Entrambi i contatori crescono nel tempo ma non in modo omogeneo perché ci sono azioni che impiegano poco tempo per essere eseguite (alcune un solo istante di tempo) ed altre che durano a lungo. Si noti che il vincolo di *maxduration* è sul tempo.
Ad ogni ciclo AGENT è a conoscenza sia di *time* che di *step* visto che status (definito nel MAIN) è visibile anche in AGENT: inoltre le percezioni emesse da ENV e ricevute da AGENT contengono anche esse l'informazione su ciclo e tempo corrente.

Il compito di incrementare questi parametri è affidato ad ENV perché si assume che il tempo di esecuzione di una azione fisica sia significativamente maggiore del tempo preso dall'agente per decidere la prossima azione da compiere. Pertanto *time* misura il tempo di execution e non il tempo di deliberation. Durante la fase di test sarà necessario verificare che l'implementazione di AGENT non violi palesemente l'ipotesi che il tempo di deliberation è piccolo (si veda il capitolo dedicato a verifica sperimentale).

MAIN

Nel modulo MAIN sono definite le strutture che sono condivise tra il modulo AGENT ed il modulo ENV. Tra le strutture condivise ci sono le percezioni.

In particolare *perc-vision* rappresenta le percezioni visive raccolte dal robot quando il robot esegue una qualunque azione. Come detto sopra, la percezione visiva copre un'area di tre celle per tre celle dove al centro (perc5) c'è la percezione relativa alla cella in cui si trova il robot, in perc1 la percezione relativa alla cella in alto a sinistra mentre in perc9 quella in basso a destra.
Le nozioni di alto e basso, destra e sinistra sono **relative alla direzione del robot**.

```
(deftemplate perc-vision
    (slot step)
    (slot time)
    (slot pos-r)
    (slot pos-c)
    (slot direction)
    (slot perc1  (allowed-values   empty debris wall gate outdoor person))
    (slot perc2  (allowed-values   empty debris wall gate outdoor person))
    (slot perc3  (allowed-values   empty debris wall gate outdoor person))
    (slot perc4  (allowed-values   empty debris wall gate outdoor person))
    (slot perc5  (allowed-values  robot empty debris wall gate outdoor person))
    (slot perc6  (allowed-values   empty debris wall gate outdoor person))
    (slot perc7  (allowed-values   empty debris wall gate outdoor person))
    (slot perc8  (allowed-values   empty debris wall gate outdoor person))
    (slot perc9  (allowed-values   empty debris wall gate outdoor person))
    )
```

perc-acoust riporta info sulla presenza di rumore nelle celle adiacenti: *yes* rappresenta la presenza di rumori provenienti da persona intrappolata tra le macerie e *no* l'assenza. La *perc-acoust* viene ricevuta dopo l'esecuzione di qualsiasi azione.

```
(deftemplate perc-acoust
            (slot step)
            (slot time)
            (slot pos-r)
            (slot pos-c)
            (slot ac (allowed-values no yes)))
```

La percezione *perc-drill* viene restituita dal modulo *ENV* solo quando al passo precedente è stata eseguita una azione di *drill*. Nello slot *result* viene restituito:

- *yes* nel caso ci sia una persona nella cella
- *no* nel caso non ci sia una persona
- *fail* quando l'azione di *drill* viene tentata su una cella in cui non c'è *debris*.

```
(deftemplate perc-drill
        (slot step)
        (slot time)
        (slot pos-r)
        (slot pos-c)
        (slot result (allowed-values yes no fail)) )
```

La percezione *perc-load* viene restituita dal modulo *ENV* solo quando al passo precedente è stata eseguita una azione di *load_debris o di unload-debris*. Nello slot *loaded* viene restituito:

- *yes* nel caso ci sia carico sul robot
- *no* nel caso non ci sia carico sul robot

```
(deftemplate perc-loaded
        (slot step)
        (slot time)
        (slot robotpos-r)
        (slot robotpos-c)
        (slot loaded  (allowed yes no)) )
```

Si noti che la percezione *perc-loaded* ritorna informazioni relative alla posizione del robot e dice se il robot è carico oppure no (l'informazione proviene da un sensore di carico). Si noti che spetta all'intelligenza del robot capire se una determinata percezione (esempio *yes*) sia congrua con l'azione svolta.

Infine per le azioni di *forward* viene restituita apposita percezione sull'esito dell'azione.
In particolare lo slot *bump* prende il valore *yes* se l'azione di *forward* è stata tentata nei confronti di una cella contenente *wall* o *debris o person*

```
(deftemplate perc-bump
        (slot step)
        (slot time)
        (slot pos-r)
        (slot pos-c)
        (slot direction)
        (slot bump (allowed-values no yes)) )
```

Altre primitive condivise tra il modulo Agent ed il modulo ENV sono le azioni. La struttura per modellare una azione è

```
((deftemplate exec
    (slot step)
    (slot action
        (allowed-values forward turnright turnleft
                drill load_debris unload_debris
                wait inform done))
    (slot param1)
    (slot param2)
    (slot param3))
```

Si noti che alcune azioni non hanno alcun bisogno dei parametri (es *turnleft*): in questo caso i valori dei parametri è NA (not applicable). Il contenuto dei parametri varia a seconda della azione considerata.
Nel caso di una azione di *inform*:

- Lo slot param1 contiene la pos-r della cella su cui si va a fare informazione
- Lo slot param2 contiene la pos-c della cella su cui si va a fare informazione
- Lo slot param3 contiene l' informazione inviata, cioè *person*, *checked* o *clear*

Nel caso *drill*, *load_debris* o *unload_debris*:

- Lo slot param1 contiene la pos-r della cella su cui si intende operare
- Lo slot param2 contiene la pos-c della cella su cui si intende operare
- Lo slot param3 è settato a NA

Infine nel modulo *MAIN* sono definite le strutture che sono destinate a contenere le info (disponibili a priori) su come è fatto l'edificio e lo stato inziale dell'agente:

```
(deftemplate init_cell
        (slot pos-r)
        (slot pos-c)
        (slot contains (allowed-values empty debris wall gate outdoor unknown)))
```

La posizione iniziale dell'agente è definita dal template
(deftemplate initial_agentposition (slot pos-r) (slot pos-c) (slot direction))
mentre si assume che all'istante iniziale il robot sia scarico.

Nel seguito verrà spiegato in che modo questa informazione viene resa nota al sistema

ENV

Il modulo ENV è molto complesso perché deve modellare l'evoluzione dell'ambiente che non è solo dovuta alle azioni decise da AGENT, ma anche dagli altri agenti (soccorritori).

La conoscenza sullo stato reale del mondo ad ogni istante di tempo è una prerogativa esclusiva di ENV (che è omnisciente): tale informazione è acquisita da *ENV* leggendo in fase di inizializzazione il file real_map.txt che contiene asserzioni sulle celle dell'ambiente istanziando il template:

```
(deftemplate real_cell
        (slot pos-r)
        (slot pos-c)
        (slot contains (allowed-values empty debris wall gate outdoor person))
        (slot injured (allowed-values yes no))
```

dove lo slot *injured* contiene *yes* oppure *no* a seconda che ci sia una persona intrappolata o meno.

Sempre in fase di inizializzazione il modulo ENV si preoccupa di creare le asserzioni relative a *init_cell* (quelle che servono come conoscenza a priori per l'agente robotico) estraendo le informazioni da quanto presente in *real_cell*. In particolare ci sarà un fatto di tipo *init_cell* per ogni cella

descritta in *real_cell* ma l'informazione sarà molto parziale (nulla sulla presenza di persone intrappolate nelle macerie e conoscenza sul tipo di cella solo su quelle outdoor e sulle celle confinanti con outdoor, per tutte le altre celle lo slot *contains* sarà impostato ad unknown).

L'informazione su quanti soccorritori sono presenti nell'ambiente e come si muovono è dinamica (inizialmente si suppone che tutti i soccorritori siano in celle outdoor) ed è di pertinenza esclusiva del modulo *ENV*. Pertanto solo *ENV* ha tale informazione esplicita (l'agente può osservare la presenza di *person* solo in modo molto parziale attraverso le *perc-vision*).

Rispetto ad *ENV* sono eventi esogeni tanto le azioni decise da AGENT (che ENV viene a conoscere visto che i fatti di tipo *exec* sono visibili in *ENV*) quanto le azioni decise da altri agenti. Queste ultime sono in qualche modellate supponendo che ENV venga a conoscenza (in certi istanti di tempo) delle decisioni dei soccorritori in termini di movimento.

ENV riceve le informazioni relative agli eventi esogeni leggendo il file hystory.txt che contiene

- Una asserzione relativa a *maxduration*
- Una asserzione relativa a *initial_agentposition*
- Eventuali asserzioni relative alla presenza di soccorritori. Questa è fornita da instanze di un template specificato in ENV

```
(deftemplate personstatus
        (slot step)
        (slot time)
        (slot ident)
        (slot pos-r)
        (slot pos-c)
        (slot activity)
        (slot move)
)
```

dove lo slot *activity* contiene *out* se il soccorritore è outdoor, *stand* se in piedi (fermo), oppure l'identificatore di un *path* se il soccorritore sta seguendo uno specifico percorso, mentre lo slot *move* specifica a quale passo del percorso intrapreso si trova il soccorritore

- Eventuali asserzioni relative alla decisione di un soccorritore di muoversi utilizzando il deftemplate

```
(deftemplate personmove
```

```
        (slot step)
        (slot ident)
        (slot path-id)
)
```

dove lo slot *step* indica a quale *step* avviene l'evento, mentre *ident* è l'identificatore del soccorritore che decide di iniziare il movimento mentre il path-id contiene l'identificatore del percorso che intende seguire.

- Eventuali asserzioni relative al cammino che il soccorritore segue espresse come fatti ordinati (si veda l'esempio nella sezione dedicata al Testing del sistema)

Si noti che le info contenute in *history.txt* devono essere fornite in modo appropriato tenendo conto sia dell'ambiente in cui si opera che delle possibili tempistiche .
In particolare le descrizioni del movimento dei soccorritori devono essere contestualizzate all'ambiente: il soccorritore non può andare in una cella occupata oppure finire in una cella inesistente.
Anche quando è stato iniziato uno spostamento il modulo ENV verifica se un passo del percorso può essere eseguito: infatti se la cella dove intende andare il soccorritore è occupata , ENV rimanda di un ciclo questa azione. Si noti che l'agente robotico paradossalmente potrebbe bloccare un soccorritore se depositasse macerie nella cella (in origine libera) in cui voleva andare il soccorritore.

Per semplificare l'implementazione del sistema si suppone che ad ogni ciclo occorra al più un *personmove*, mentre nello stesso ciclo più soccorritori si possono muovere in contemporanea.

Il modulo ENV utilizza inoltre un gran numero di informazioni (e di strutture) per gestire l'evoluzione del mondo. Anche se queste non riguardano direttamente il modulo AGENT, per lo studente è molto utile analizzarne alcune per capire come viene gestito il complesso meccanismo di sincronizzazione tra attività tra diversi agenti (in particolare di quello robotico).

AGENT

Il modulo AGENT è fornito in una versione assolutamente embrionale perché lo scopo del progetto è proprio lo sviluppo di questo modulo. Nella versione fornita esso definisce solo l'interfaccia di comunicazione (fatti di tipo *exec*) e ad ogni ciclo deve essere l'utente che fornisce l'azione che il robot deve eseguire.

Anche se la conoscenza di base su come è fatto l'ambiente è condivisa (l'agente robotico ha accesso a questa informazione), nel MAIN è definita solo la struttura inziale dell'ambiente. Pertanto AGENT non deve operare su questa struttura condivisa cosi come sullo stato iniziale dell'agente.

Per questa ragione nella versione embrionale dell'AGENT sono presenti due regole (*beginagent_kcell_no_injured* e *beginagent_kcell_maybe_injured*) che hanno il compito di "copiare" in strutture interne di AGENT le informazioni relative alla mappa e alla stato inziale dell'agente. Le strutture (è solo una proposta) sono rappresentate dai seguenti template:

```
(deftemplate K-cell
    (slot pos-r)
    (slot pos-c)
    (slot contains (allowed-values empty debris wall gate outdoor person unknown))
    (slot injured (allowed-values yes no unknown))

(deftemplate K-agent
    (slot step)
    (slot time)
    (slot pos-r)
    (slot pos-c)
    (slot direction)
    (slot loaded))
```

Nell'implementazione di AGENT è consigliabile seguire una strategia di sviluppo incrementale che parta dalle funzionalità di base dell'agente stesso. In particolare si suggerisce di partire con

- l'implementazione delle regole che devono fornire informazioni al COAS per mezzo di *inform*
- l'implementazione di una strategia che decide quale cella andare a visitare e eventualmente eseguire una *drill*
- l'implementazione di una strategia di path planning per pianificare il percorso dell'agente robotico da una cella all'altra
- l'implementazione di una strategia che non intraprenda una azione se non c'è tempo sufficiente per completarla (non sforare *maxduration*)

Si consiglia inoltre di sviluppare una versione base dell'agente che sia in grado di eseguire correttamente l'analisi di una zona della mappa senza porsi subito l'obiettivo di ottimizzare le operazioni di carico e scarico. Queste (come la capacità dell'agente di pianificare percorsi in presenza di soccorritori) è da sviluppare dopo aver sviluppato le funzionalità di base.

E’ sicuramente utile che si analizzi con una certa cura cosa fa il modulo ENV in modo da comprendere bene il comportamento dell’ENV specie nel caso in cui AGENT non sia ancora in grado di scegliere al meglio le azioni da eseguire (per cui esegue azioni sbagliate e/o inutili con conseguente accumulo di un gran numero di penalità).

LA SPERIMENTAZIONE

Come detto sopra, le prestazioni di AGENT dipendono da molti fattori:

- le strategie adottate da AGENT
- la dimensione e la struttura dell’ambiente
- il numero e la dislocazione delle macerie
- il numero e la dislocazione delle persone intrappolate

Ovviamente le strategie adottate da AGENT giocano un ruolo cruciale, ad esempio non mandare *inform* al COAS o mandare informazioni sbagliate porta a penalità inaccettabili.

Può essere utile implementare alcune strategie (anche un paio potrebbe essere accettabile se il gruppo di lavoro è ristretto) di diversa complessità per poter valutare se si riscontrano variazioni significative di prestazioni.

Si raccomanda di provare il sistema (i sistemi se le strategie sono più di una) in domini a diversi livelli di complessità (auspicabile che i diversi gruppi di studenti arrivino a formulare una serie di ambienti condivisi in modo che il sistema venga valutato sullo stesso insieme di domini).
Per iniziare questa opera di condivisione la rappresentazione grafica di un ambiente è riportata nel seguito.

10	out	out	out	out	out	out	out	out	out	out	Out
9	out	w	w	w	w	w	gate	debris	debris	w	Out
8	out	w			w				debris/P	w	Out
7	out	debris	debris		w			debris/P		gate	Out
6	out	debris			w		debris/P	debris		w	Out
5	out	w		w	w		w	w	w	w	Out
4	out	w		debris			debris	debris/P		w	Out
3	out	w		debris/P						w	Out
2	out	w	w	w	gate	w	w	w	gate	w	Out
1	out	out	out	out	out	out	out	out	out	out	Out
	1	2	3	4	5	6	7	8	9	10	11

La descrizione in termini di fatti CLIPS dell'ambiente sopra riportato è contenuta nel file RealMap-20.txt. Per far girare il sistema, il file in questione deve essere copiato nel file InitMap.txt perché ENV legge sempre da InitMap.txt

Per le informazioni relative a un particolare scenario relativo al dominio in questione, è necessario copiare nel file history.txt il contenuto del file history-300-2p.txt

L' informazione sulla durata massima della missione è data dal fatto

(maxduration 20)

mentre questo fatto specifica la posizione iniziale dell'agente robotico

(initial_agentposition (pos-r 2) (pos-c 5) (direction north))

I fatti

(personstatus (step 0) (time 0) (ident C1) (pos-r 1) (pos-c 9) (activity out))

(personstatus (step 0) (time 0) (ident C2) (pos-r 10) (pos-c 5) (activity out))

indicano che in questo scenario ci sono due soccorritori (il cui ident è C1 e C2) che a step 0 e time 0 si trovano nelle celle (1,9) e (10,5) e la loro activity è out (sono fuori dell'edificio e lo stanno osservando)

Il fatto

(personmove (step 5) (ident C1) (path-id P1))

indica ad ENV che il soccorritore C1 al ciclo 5 decide di passare da activity *out* ad activity *P1* cioè di iniziare il movimento descritto come P1

Il fatto

(personmove (step 19) (ident C1) (path-id P2))

Indica che al passo 19 il soccorritore C1 inizia una nuova attività (cioè passa da activity *stand* a activity *P2* – identificatore del nuovo cammino intrapreso).

I seguenti fatti specificano quali sono le mosse compiute da C1 nei diversi passi del cammino P1 (composta da 5 mosse in cui gli ultimi due parametri indicano le coordinate della cella in cui il soccorritore va con quella mossa)

(move-path P1 1 C1 2 9)
(move-path P1 2 C1 3 9)
(move-path P1 3 C1 3 8)
(move-path P1 4 C1 3 7)
(move-path P1 5 C1 3 6)

mentre la descrizione del secondo cammino (identificato con P2) è catturata dai seguenti fatti

(move-path P2 1 C1 4 6)
(move-path P2 2 C1 4 5)
(move-path P2 3 C1 3 5)
(move-path P2 4 C1 2 5)
(move-path P2 5 C1 1 5)
(move-path P2 6 C1 1 4)