NavigAItors

Progetto di Robotica e Intelligenza artificiale 2

Agostino Messina, Alberto Conti, Federico Princiotta Cariddi.

Indice

- Descrizione del progetto
- Ambiente di sviluppo
- Agenti intelligenti coinvolti
- Filtri di Kalman
- Algoritmo A*
- Integrazione del modello BDI
- Ontologia
- Protégé
- Comunicazione tra agenti

Descrizione del progetto.

Abbiamo realizzato il progetto NavigAItors con lo scopo di simulare una missione di salvataggio di ostaggi in cui un robot intelligente, supportato da un drone, analizza un ambiente ostile per trovare il percorso ottimale e raggiungere gli ostaggi detenuti da un nemico.

Il drone mappa l'area dall'alto, mentre il robot esegue il salvataggio, coordinato da algoritmi avanzati per massimizzare efficienza e sicurezza, aumentando le probabilità di successo della missione.

Ambiente di sviluppo.

Lo sviluppo dell'intero progetto è stato portato avanti in Unity e Visual Studio Code.

Agenti intelligenti coinvolti.

Robot di terra

Drone

Enemy (solo per AI)

Drone - Riepilogo.

- **Funzioni principali**: Identifica nemici con precisione grazie all'applicazione dei filtri di Kalman e effettua una mappatura dell'ambiente.
- **Apprendimento**: Durante il training con ML-Agents, il drone è stato esposto a scenari casuali con nemici ed edifici disposti in modo variabile. Questo ha permesso al drone di affrontare situazioni sempre diverse, migliorando le sue capacità di navigazione e decisione.
- **Input e output**: la rete neurale utilizza come input la posizione e rotazione del drone, le sue velocità, il conteggio dei nemici trovati, la distanza della zona inesplorata più prossima e la percentuale di esplorazione totale. L'output, invece, corrisponde ai movimenti lungo i tre assi e le rotazioni di yaw, pitch e roll.
- **Rewards**: Il training si basa sul reinforcement learning, il drone riceve feedback immediato in base alle sue azioni. Ad esempio, evitare un ostacolo o identificare un nemico genera un reward positivo, mentre collisioni o mancati rilevamenti portano a penalità. Questo sistema di reward/punishment guida il drone a ottimizzare autonomamente le sue decisioni.

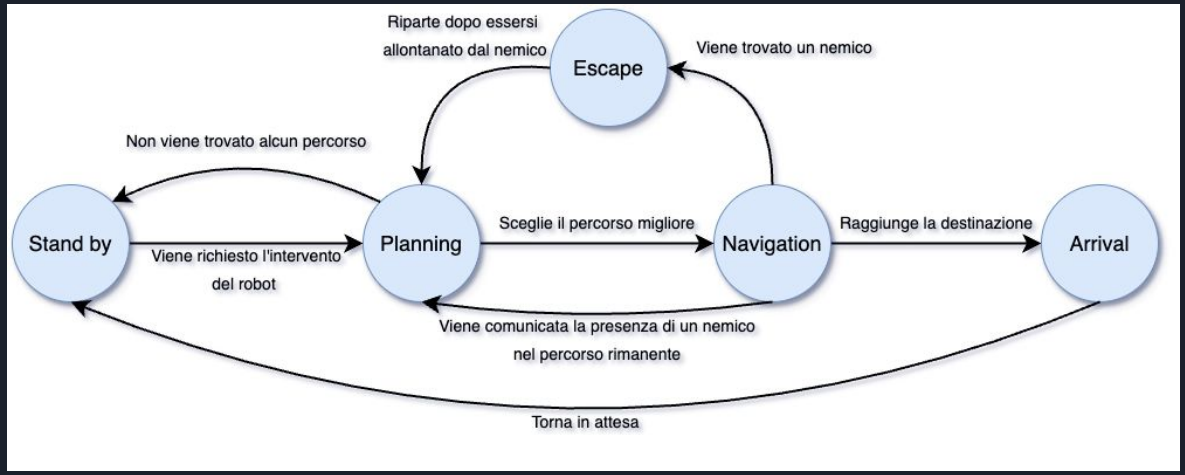
Robot di terra - Riepilogo.

- **Capacità e funzionalità**: il robot è composto da diversi stati con una FSM. Può navigare autonomamente verso la destinazione impostata, rilevare nemici e ostacoli dinamici e adattare il suo percorso in risposta a cambiamenti dell'ambiente.

- **Rilevamento e navigazione**: Il robot utilizza dei sensori a 360° per identificare i nemici vicini, aggiornare la griglia e con l'algoritmo A* rigenerare il percorso per evitare collisioni.

- **Interazione con il drone**: quando il drone esplora l'ambiente, aggiorna in tempo reale la griglia con informazioni cruciali. Il robot accede a questi dati aggiornati per pianificare e adattare il proprio percorso in modo da ottimizzare la navigazione.

Robot di terra - Automa a stati finiti.



La logica del robot di terra prevede di operare con le modalità di un Automa a Stati Finiti (FSM) suddividendo le proprie azioni in base ai differenti stati in cui esso può entrare.

Filtri di Kalman.

Il Filtro di Kalman è stato utilizzato per affinare la precisione della localizzazione dei nemici, migliorando così la capacità del sistema di simulazione di rispondere in modo accurato e tempestivo alle minacce. Questo algoritmo è cruciale per ottimizzare le stime dello stato dei nemici basate su misurazioni imprecise.

Il Filtro di Kalman è un algoritmo che stima la posizione o lo stato di un oggetto, anche in presenza di incertezze. Funziona attraverso un ciclo continuo di due fasi: **predizione** e **correzione**.

1. Nella fase di predizione, si calcola una stima della posizione futura basandosi su un modello teorico, come il movimento precedente. Tuttavia, questa stima non è perfetta perché può essere influenzata da errori.
2. Nella fase di correzione, il filtro confronta la stima con dati reali, come misurazioni da sensori, per correggere eventuali discrepanze. Anche questi dati, però, possono contenere rumore o imprecisioni.

Il Filtro di Kalman bilancia in modo ottimale la predizione e la misurazione per produrre una stima più accurata. Questo ciclo si ripete continuamente, permettendo al filtro di adattarsi in tempo reale e ridurre l'incertezza.

Kalman_filter $(\mu_{t-1}, \Sigma_{t-1}, u_t, z_t)$:

Predizione:

$$\bar{\mu}_t = A_t \mu_{t-1} + B_t u_t$$

$$\bar{\Sigma}_t = A_t \Sigma_{t-1} A_t^T + R_t$$

Correzione:

$$K_t = \bar{\Sigma}_t C_t^T \left(C_t \bar{\Sigma}_t C_t^T + Q_t \right)^{-1}$$

$$\mu_t = \bar{\mu}_t + K_t \left(z_t - C_t \bar{\mu}_t \right)$$

$$\Sigma_t = \left(I - K_t C_t \right) \bar{\Sigma}_t$$

return μ_t, Σ_t

Algoritmo A*

L'algoritmo A* serve al robot per calcolare il percorso ottimale per il raggiungimento del goal combinando costi di movimento ed euristiche. Parte dalla posizione iniziale, esplora le celle valutando il costo totale e aggiorna dinamicamente il percorso fino alla destinazione.

Integrato con il GridManager, permette al robot di adattarsi a variazioni dell'ambiente, garantendo efficienza e sicurezza. Le fasi nel dettaglio sono:

- **Inizializzazione**: Partendo dalla cella iniziale, l'algoritmo aggiunge celle alla lista aperta e le sposta nella lista chiusa man mano che vengono esplorate.
- **Calcolo dei Costi**: Ogni cella ha un costo di movimento associato e un costo euristico che stima la distanza dalla cella di destinazione.
- **Ottimizzazione del Percorso**: L'algoritmo cerca il percorso con il costo totale minimo, aggiornando i percorsi potenziali con nuove scoperte fino a raggiungere la destinazione.

$$f(n) = g(n) + h(n)$$

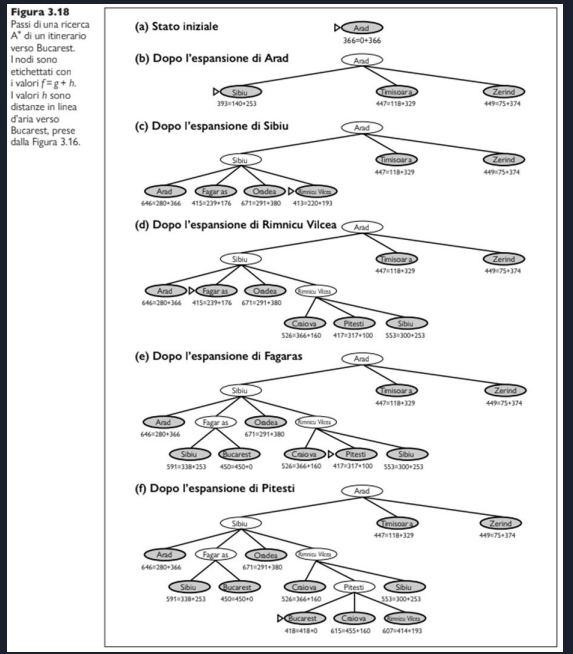
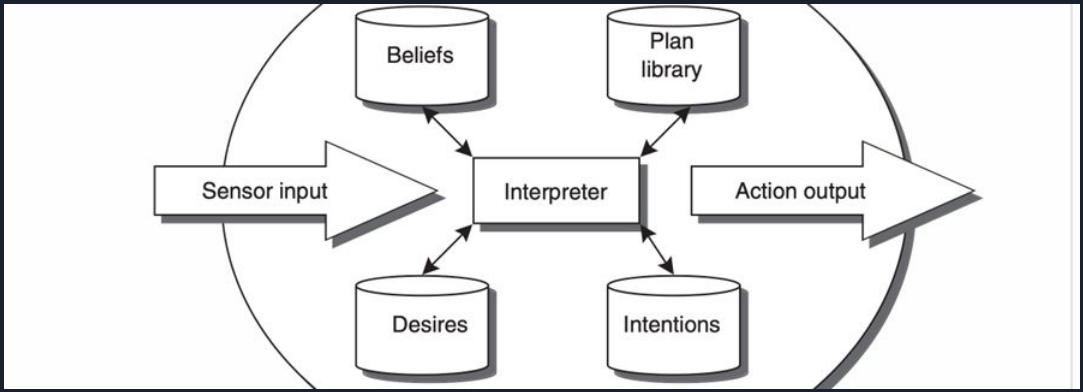


Figura 3.18 Passi di una ricerca A* di un itinerario verso Bucarest. I nodi sono etichettati con i valori $f = g + h$. I valori h sono distanze in linea d'aria verso Bucarest, prese dalla Figura 3.16.

Integrazione del modello BDI.

Il modello BDI (Belief-Desire-Intention) viene applicato nel progetto NavigAItors per gestire agenti autonomi in un ambiente complesso. Ogni agente segue il modello BDI, il quale che permette loro di prendere decisioni in modo autonomo basandosi sulla conoscenza dell'ambiente circostante.

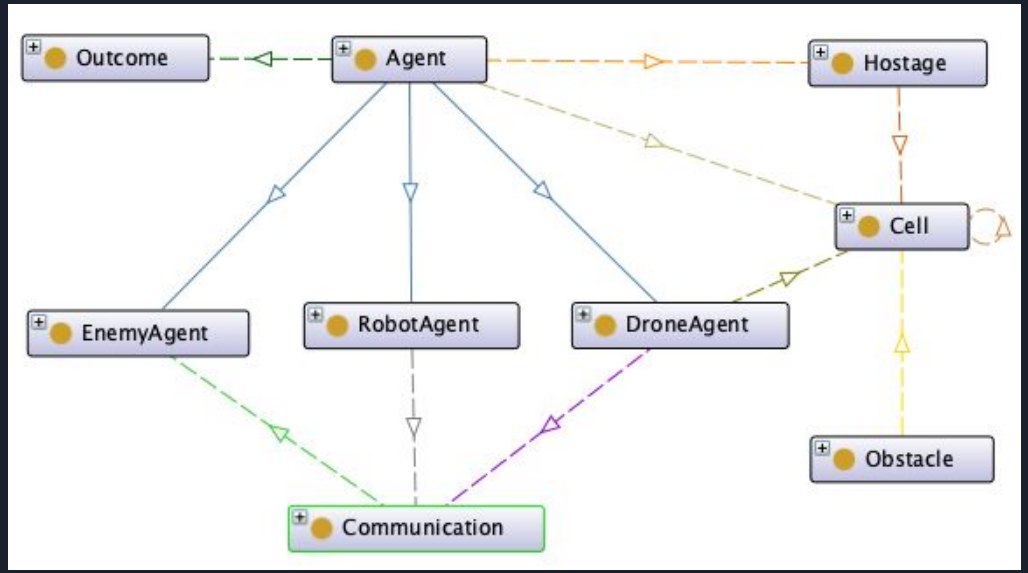
- **Beliefs**: Gli agenti memorizzano e aggiornano informazioni come la posizione di ostaggi, nemici e ostacoli.
- **Desires**: Il Robot di Terra mira a salvare gli ostaggi, mentre il Drone rintraccia il nemico e il Nemico cerca di ostacolare il salvataggio.
- **Intentions**: Il Robot di Terra usa l'algoritmo A* per pianificare il percorso, il Drone aggiorna in tempo reale la posizione del nemico e il Nemico si muove strategicamente per intercettare il Robot.



Ontologia

L'ontologia funge da "mappa concettuale" dell'ambiente. Essa ci fornisce una struttura formale per rappresentare la conoscenza condivisa tra gli agenti BDI (Drone, Robot di Terra, Nemico).

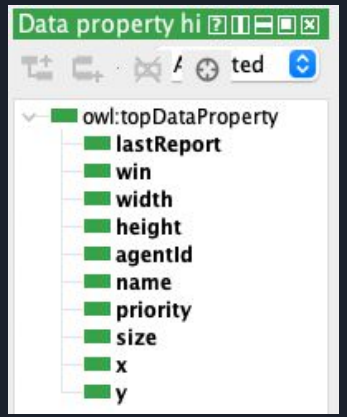
Abbiamo utilizzato l'editor di ontologie Protégé per definire classi, proprietà, gerarchie e istanze e successivamente abbiamo proseguito con l'integrazione con il Knowledge Graph Neo4j per la gestione e l'aggiornamento delle informazioni.

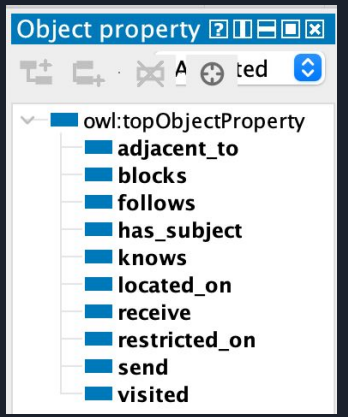


Protégé

Protégé permette di rappresentare anche le proprietà degli oggetti (relazioni) e le proprietà dei dati (attributi) e come gli elementi interagiscono tra loro.
È stato dunque usato per mantenere una visione d'insieme di tutti i concetti e le relazioni, nel dettaglio:

- Creare e gestire le classi e le loro gerarchie
- Definire e documentare le proprietà (object e data) che descrivono le relazioni e gli attributi.
- Verificare la consistenza dell'ontologia tramite reasoner, assicurando che non esistano contraddizioni nella definizione dei concetti.
- Esportare l'ontologia in formati standard (OWL/RDF) per facilitarne l'integrazione con altri sistemi (Neo4J).

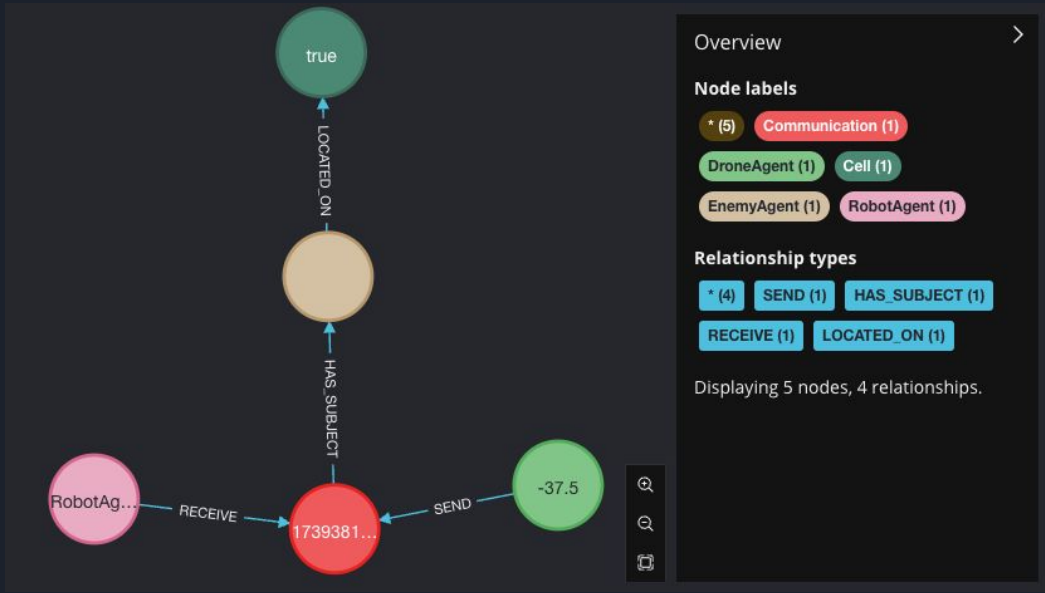




Caso d'uso - Comunicazione tra agenti

Abbiamo implementato la comunicazione tra il robot di terra e il drone sfruttando il Knowledge Graph Neo4j il quale descrive nel dettaglio come avviene questa comunicazione.

Gli agenti BDI consultano il grafo per leggere o aggiornare le informazioni. La classe communication ha come scopo quello di registrare l'informazione scambiata e tenere traccia di chi l'ha inviata e chi l'ha ricevuta.




```
FUNCTION GetEnemyPosition():
  CONNECT to Neo4j

  //1 Trova (r: RobotAgent) che riceve (RECEIVE) un Communication
(comm)
  // il quale è inviato (SENDS) da un DroneAgent
    MATCH
(r:RobotAgent)-[:RECEIVE]->(comm:Communication)<-[:SENDS]-(:D
roneAgent)

  //2 comm ha come messaggio (HAS_SUBJECT) l'EnemyAgent (e)
    MATCH (comm)-[:HAS_SUBJECT]->(e:EnemyAgent)

  //3 L'EnemyAgent e si trova su una cella (c:Cell) tramite
LOCATED_ON
    MATCH (e)-[:LOCATED_ON]-(c:Cell)

  //4 Restituisce la posizione della cella
    RETURN c.x AS x, c.z AS z
```

Drone - Riepilogo AI.

Il drone svolge un ruolo di supporto nei confronti del Robot di Terra attraverso l'esplorazione dell'area, il rilevamento del nemico e la comunicazione delle informazioni strategiche. Grazie ai suoi sensori simulati e alla connessione con Neo4j, il drone garantisce un flusso costante di dati aggiornati per ottimizzare le decisioni operative. Nel dettaglio:

Pattugliamento: Si muove in modo ragionato nelle aree non ancora visitate per coprire l'intera mappa. Se individua il nemico, interrompe il pattugliamento e inizia a seguirlo. Durante l'esplorazione, aggiorna continuamente la sua Knowledge Base (KB).

Rilevamento del Nemico: Utilizza sensori simulati per monitorare l'ambiente e rilevare la presenza del nemico entro un certo raggio d'azione. Se lo individua, ne traccia i movimenti.

Comunicazione con il Robot di Terra: Registra in tempo reale la posizione del nemico nella sua KB su Neo4j. Il Robot di Terra consulta periodicamente queste informazioni per ottimizzare il suo percorso ed evitare il nemico, migliorando l'efficienza della missione.

```
FUNCTION ReportEnemyPosition(droneId,
enemyPos):
 CONNECT to Neo4j
 //1 Trova (d:DroneAgent {id: $droneId})
 //2 Trova (comm:Communication {id:$commId})
 //3 Setta comm.lastReport timestamp
 //4 Collega DroneAgent a Communication (SEND)
 //5 Collega Communication a RobotAgent
(RECEIVE)
 //6 Collega Communication a EnemyAgent
(HAS_SUBJECT)
 //7) Aggiorna la cella su cui si trova l'enemy
(LOCATED_ON)
```

Robot di terra - Riepilogo AI.

Il Robot di Terra ha il compito principale di salvare gli ostaggi, evitando al contempo il nemico per completare con successo la missione. Nel dettaglio:

- **Salvataggio degli Ostaggi**: Identifica la posizione degli ostaggi e pianifica il percorso ottimale per il salvataggio, dando priorità a quello più sicuro.

- **Intercettazione del Nemico**: Se la posizione del nemico, comunicata dal drone, si avvicina troppo, il robot ricalcola il percorso per evitarlo e raggiungere comunque l'obiettivo in sicurezza.

- **Game Over**: Se il nemico intercetta il robot prima che la missione sia completata, il gioco termina con un fallimento.

```
FUNCTION FreeHostage(pos):
 CONNECT to Neo4j
 //1 Trova l'ostaggio associato alla cella
(c:Cell) nelle coordinate pos.x, pos.z
    MATCH
(e:RobotAgent)-[k:KNOWS]->(h:Hostage)-[r:RE
STRICTED_ON]->(c:Cell {x:pos.x, z:pos.z})
 //2 Rimuove la relazione r
    DELETE r
```

Enemy - Riepilogo

Il Nemico ostacola la missione del Robot di Terra, muovendosi strategicamente per controllare gli ostaggi ed evitarne il salvataggio.

- **Movimento e Strategia**: Si sposta tra gli ostaggi seguendo un ordine di priorità. Se un ostaggio viene liberato, lo rimuove dalla sua base di conoscenza e sorveglia quelli che non sono ancora stati salvati.

- **Intercettazione del Robot di Terra**: Se individua il robot, si dirige verso l'ostaggio più vicino per impedirne il salvataggio. Se riesce ad avvicinarsi abbastanza, può bloccare il robot e terminare la simulazione con un fallimento.

```
MATCH (a:  EnemyAgent)-[:KNOWS]->(h: Hostage)
RETURN h.x AS x, h.z AS z ORDER BY h.priority DESC
```

```
MERGE (o:Outcome {id: 'GameOutcome'})
 SET o.win = false

 WITH o
 MATCH (r:RobotAgent), (e:EnemyAgent),(d:DroneAgent)
 MERGE (r)-[:FOLLOW]->(o)
 MERGE (e)-[:FOLLOW]->(o)
 MERGE (d)-[:FOLLOW]->(o)
```

Grazie per l'attenzione!