ROBOT WARS

Corso di Sistemi Autonomi

A.A. 2015/2016

Matteo Delvecchio

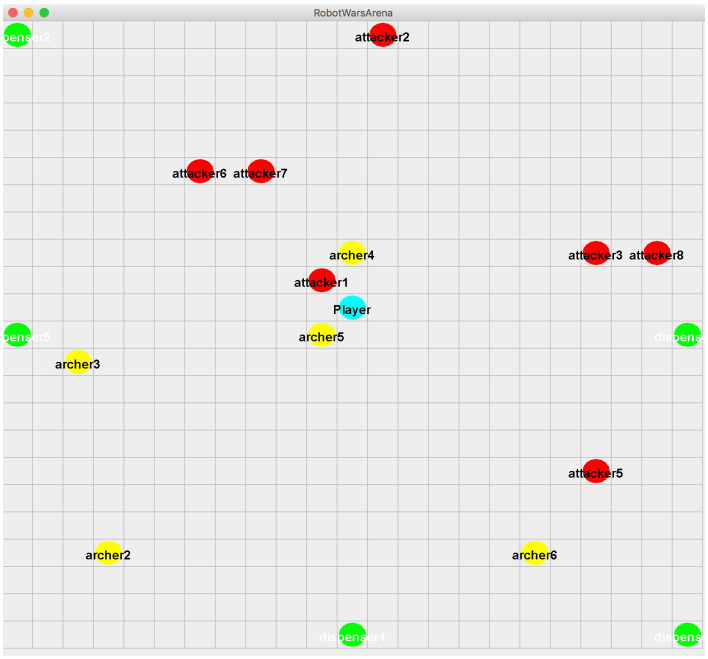
matteo.delvecchio4@studio.unibo.it

Introduzione al Progetto e panoramica degli elementi costitutivi

Robot Wars è un gioco in cui l'utente è chiamato a configurare alcuni aspetti di un robot detto "Player" al fine di sconfiggere, possibilmente nel minor tempo, una schiera di robot più deboli, eterogenei.
Il gioco è costituito da un'arena quadrata formata da 23 caselle in cui si trovano 8 robot detti "Attacker" , 6 robot "Archer", 5 robot "Dispenser" e naturalmente il robot dell'utente. Non vi sono altre interazioni durante la partita oltre alla configurazione iniziale.
Il gioco termina quando il robot del giocatore è riuscito ad eliminare tutti gli altri robot dotati di capacità offensive oppure quando viene distrutto. A fine partita è mostrato il tempo di gioco e, in caso di vittoria, eventuali punti bonus collezionati al fine di migliorare il risultato.

Environment

Il campo di gioco, detto ARENA è costituito da una griglia di 23x23 caselle. Altre informazioni reperibili oltre a quelle relative alle posizioni dei robot sono costituite dalla presenza di muri ed angoli ossia i confini del campo di gioco



Player Robot

Robot configurabile dall'utente. Il suo tipo di spostamento sulla grigia è di tipo diagonale.
Le sue caratteristiche iniziali (nella versione base del gioco) sono:
VITA = 300
LATENZA = 360
ATTACCO = 30
Mediante configurazione si possono assegnare fino a 6 punti caratteristica, tuttavia a ciascuna caratteristica può essere assegnato fino ad un massimo di 3 punti caratteristica. Inoltre è necessario specificare l'arma con cui il robot entrerà nell'arena.
Ogni singolo punto caratteristica conferisce al Player uno dei seguenti benefici:

+50 VITA
-60 LATENZA
+5 ATTACCO

Per quanto riguarda le armi, in questa demo la scelta possibile è di tre tipologie di armi:

Spiked chain (mazza chiodata) → Ogni volta che nelle immediate vicinanze è presente un nemico viene portato un attacco radiale che colpirà anche eventuali altri robot, ma con una potenza del 60%.
Bow (Arco) → Ogni volta che viene individuato un nemico in lontananza, esso viene colpito, tuttavia i nemici vicini non possono essere colpiti.
Sword (Spada) → Colpisce il primo nemico individuato nelle immediate vicinanze, ma non è in grado di colpire i nemici distanti

Nel file ***robotWarsStable.mas2j*** è possibile configurare il Player, tenendo a mente che configurazioni non valide bloccheranno l'avvio del gioco. Riportiamo un esempio di configurazione valida al fine di associare la semantica alla sintassi di configurazione:

```
environment: env.ArenaEnv(gui,2,2,2,bow)
```

Escludendo il parametro "*gui*", necessario e non modificabile, abbiamo come secondo parametro il numero di punti caratteristica associati alla vita, a seguire quelli associati alla latenza e per ultimi quelli associati all'attacco. Chiude la configurazione il tipo di arma (*bow*, *sword*, *spiked_chain*). Tale configurazione è considerata accettabile in quanto nessuna caratteristica riceve più di tre punti e la somma dei punti così distribuiti non supera le 6 unità.

Archer Robot

Robot in grado di colpire sia a distanza che a contatto, caratterizzato da movimenti diagonali. Le sue caratteristiche sono fisse:
VITA = 90
LATENZA = 390
ATTACCO = 15
PROIETTILI DI DILITIO = 8
Ogni colpo consuma un proiettile di DILITIO. In caso di esaurimento colpi è necessario fare rifornimento presso un **Dispenser Robot**.

Dispenser Robot

Robot la cui funzione è principalmente di supporto ma come effetto secondario spesso produce intralcio in sfavore del **Player Robo**t. Le sue caratteristiche iniziali sono fisse:
VITA = 190
LATENZA = 600
ATTACCO = 0
RISERVA DI DILITIO = 20
Esso è l'unico robot incapace di muoversi. La sua funzione è però fondamentale per poter ricaricare gli **Archer Robo**t. Ogni rifornimento comporta il trasferimento di 5 munizioni agli Archer. La sua seconda funzione è quella di tenere occupato il Player dando possibilità di fuga ad eventuali robot in pericolo.

Attacker Robot

Robot che colpisce solo in maniera ravvicinata, il suo movimento all'interno dell'arena è detto a tappeto, ovvero si muove in orizzontale sulla griglia e, a seconda del caso, percorre orizzontalmente la riga superiore o inferiore, in senso opposto. Questo movimento è perpetuato in modo da percorrere tutta la griglia. Le sue caratteristiche iniziali sono fisse:
VITA = 80
LATENZA = 450
ATTACCO = 18

Percepts

Attraverso le percezioni i robot sono in grado di ricavare i seguenti dati:
myPosition —> Posizione attuale del robot sotto forma di coordinate cartesiane.

(Insieme di) **nearMe**—> Posizione di un robot vicino, percepito a partire da tre caselle di distanza, riportante info di natura cartesiana e di identità e tipologia esatta del robot percepito.

wall —> vicinanza di un muro con il relativo valore numerico di identificazione:

w2

w1 w-1

w-2

corner—> vicinanza di un angolo con il relativo valore numerico di identificazione:

d1 d2
X
d3 d4

Knowlege

Oltre alle informazioni che costituiscono le caratteristiche del robot, la base di conoscenza si amplia di informazioni ricavate del robot come:

wander—> Nel caso di un robot offensivo indicano la prossima mossa che il robot sa che dovrebbe fare dato il tipo di moto che lo caratterizza.

direction—> Credenza ricavata dal sotto goal *calculate_direction*. La sua sintassi è (D,V) in cui D è il tipo di direzione (D={a;d}) e V è il valore di tale direzione (ovvero il valore numerico di angolo o muro verso il quale il robot deve dirigersi),

Actions

Le azioni che ogni robot offensivo può compiere sono:

wanderMove—> Spostamento sulla griglia di gioco. In caso sia di tipo diagonale la sua sintassi è (d,N). N={1;2;3;4} indica la diagonale in cui ci si sta direzionando. In caso sia altro tipo la sua sintassi è (a,M,N).
M={-2;2} indica il senso globale di percorrenza, in relazione al muro (-2=discendente; 2=ascendente).
N={-1;1; -2; 2} indica il muro verso il quale ci si sta dirigendo nell'istante attuale.

calculate_direction—> Permette il calcolo della direzione in cui si trova un robot vicino. Viene utilizzato dai robot come sotto goal durante una *wanderMove* condizionata dalla presenza di un robot che si rivela rilevante.

follow—> Spostamento in direzione di un robot, sfrutta il risultato dell'azione *calculate_direction*.

runAway—> Permette di allontanarsi dal robot in questione, dopo averne calcolato la direzione.

dead—> Notifica di morte verso gli altri robot.
———————————
Il Player, dispone inoltre di azioni specifiche per le armi di cui dispone: ***hit_around***, ***hit_one_far*** e ***hit_one_near***.

Interazione mediante messaggi

Attraverso scambio di messaggi i robot sono in grado di fornire diversi tipi di informazioni:

hit—> Accompagnato da un valore numerico che indica l'entità del danno, indica che il robot è stato danneggiato.

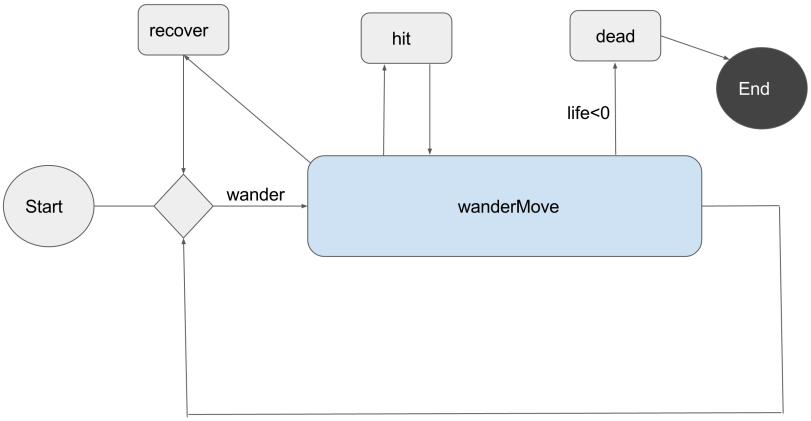
dead—> Accompagnato dall'identificativo del robot deceduto, indica ai riceventi che tale robot non esiste più.
———————————
L'interazione tra Dispenser e Archer avviene in maniera diretta:

giveMeBullets —> Accompagnato dal numero di munizioni richiesto e dall'identificativo del richiedente, permette agli Archer di chiedere ai Dispenser un rifornimento.

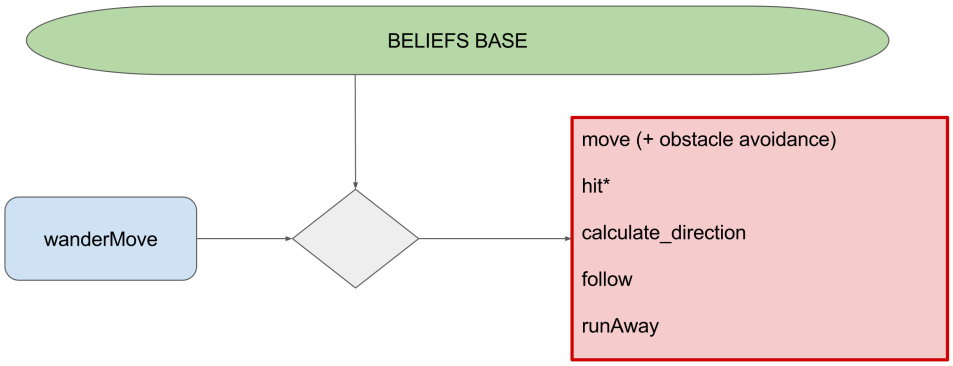
bulletSupply —> In caso di rifornimento possibile, notifica l'Archer Robot richiedente dell'avvenuto rifornimento.
nope —> Indica all'Archer che ha richiesto rifornimento l'impossibilità del Dispenser di esaudire tale richiesta.
———————————
Il Player notifica mediante un messaggio di tipo *broadcast* la sua dipartita attraverso l'informazione **lose.**

Comportamento in sintesi

Lo schema riportato presenta, in maniera generale, il funzionamento tipico dei robot dotati di capacità offensive (*Player*, *Archer*, *Attacker*). Non essendo stata specificata una semantica adeguata alla sintassi proposta, si fa affidamento al codice sorgente degli agenti.



Lo schema è comune ai robot offensivi in quanto ciò che li caratterizza è la *plan library* di esecuzione della wanderMove (oltre che le caratteristiche di tipologia di movimento, di attacco, e le statistiche sopra menzionate), sia in termini di contesti di applicabilità di tali piani, sia per i sotto ragionamenti che vengono svolti. Possiamo astrarre il funzionamento dei piani di esecuzione della wanderMove nello schema sotto riportato:



La programmazione degli agenti è eseguita utilizzando **JASON**, estensione di **AgentSpeak**, che è dunque un linguaggio sia di programmazione che di modellazione di agenti secondo paradigma BDI. Esso è in grado di modellare il **Multy-Agent System** in termini di struttura e comportamento locali delle diverse entità, di interazioni tra le entità e con l'ambiente, e do modellare l'ambiente stesso, pertanto per una dettagliata descrizione del sistema in tutte le coordinate di analisi, si rimanda al codice sorgente, che si rivela essere autoesplicativo.

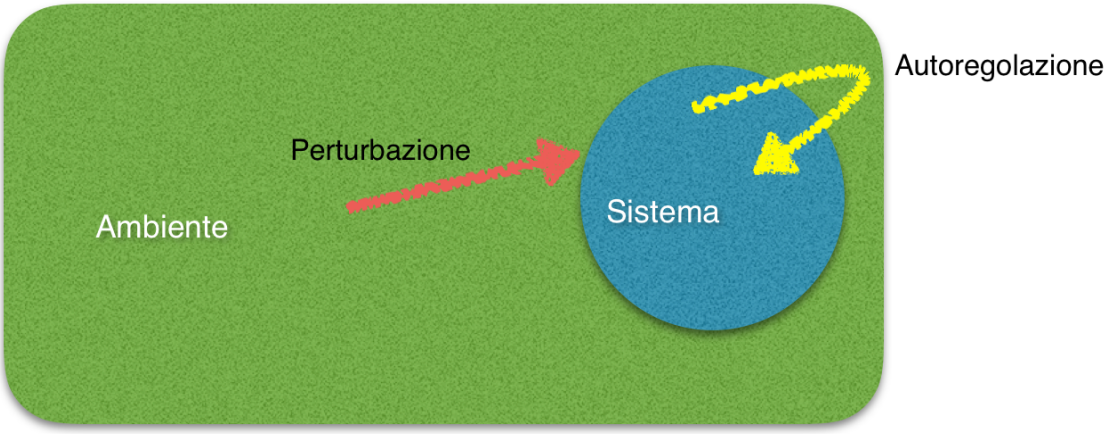
Considerazioni inerenti ai concetti di Autonomia

Il focus di questa parte di trattazione è osservare come il progetto proposto e il paradigma utilizzato mettano in rilievo i concetti visti nel corso di Sistemi Autonomi, come essi si concretizzino nelle tecnologie utilizzate e nel dominio di questo caso di studio.

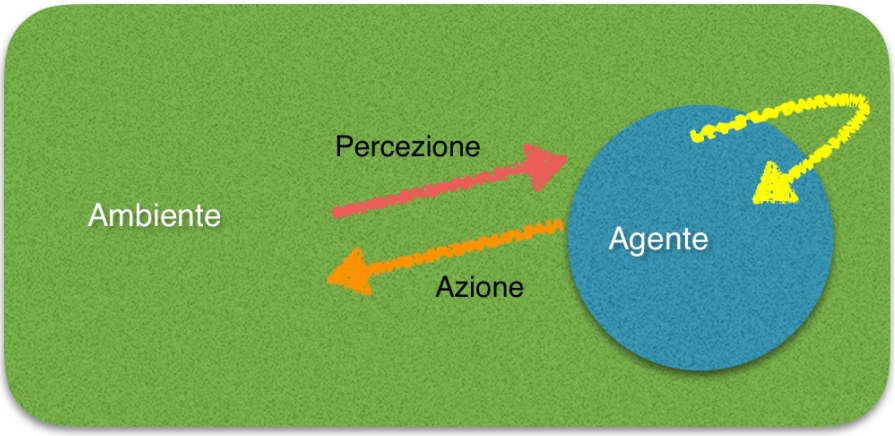
Autonomia Biologica

Partendo dalle indagini sull'autonomia in natura e sulle caratteristiche riscontrate in tale ambito, notiamo che le tali proprietà fondanti sono presenti all' intero di questo caso di studio.
Prendendo in esame **[Tho10]**, uno dei punti centrali per la definizione di autonomia è il concetto di *confine* (***boundary***), come elemento di identità del sistema. Troviamo un concreto parallelismo con questo concetto nel modello ad agenti: un agente è per definizione un oggetto il cui flusso di controllo non esce mai dai sui confini, ma solo il flusso di conoscenza, dati ed informazioni.
Tali flussi introducono un'altra tematica rilevante, la quale trae sempre ispirazione dal mondo naturale: così come i sistemi viventi non sono indipendenti dall'ambiente, allo stesso modo gli agenti di un MAS sono fortemente accoppiati con l'ambiente in cui vivono, al contesto (***situatedness***), e scambiano con esso informazioni.
Un altro concetto che emerge dal progetto proposto, fortemente correlato alla *situatedness*, e trattato in **[Kit02]** è il concetto di **robustezza**; questa caratteristica è sia un prerequisito dell'autonomia, in quanto consente altre proprietà caratterizzanti l'autonomia, che un una parte ben definita dell'autonomia, a garanzia del mantenimento di identità, struttura ed organizzazione.
La robustezza emerge nel lavoro svolto attraverso un uso di molteplici piani, in modo da far fronte al maggior numero possibile di perturbazioni e cambiamenti, oltre che la presenza di piani di *recover* per garantire la possibilità di auto ripristino in presenza di situazioni anomale. Ciò è possibile, così come in natura, attraverso strategie quali la ridondanza e sistemi di feedback.

Un concetto che abbraccia in maniera stretta il paradigma multi-agente con l'autonomia di ispirazione biologica è il principio dell'**omeostasi**; essa è la capacità di un sistema di regolare le sue condizioni interne e dunque rimanere stabile (in relazione ad alcune funzionalità).



Tale schema è assimilabile allo schema di funzionamento di un agente (che ne fa da soprainsieme delle funzionalità):



Ove Processo decisionale >= Autoregolazione ; Percezione == perturbazione

Per citare alcuni esempi di questa tipologia, abbiamo diverse situazioni in cui i robot regolano la loro traiettoria in base a condizioni ambientali e dati relativi all'ambiente circostante come la presenza di muri e di altri robot.

Così come le entità viventi stabiliscono i loro cicli vitali in funzione del tempo, allo stesso modo i robot presenti nel sistema, affrontano ciclicamente delle fasi di movimento, fuga, avvicinamento.

Autonomia generale [Ros14]

Dopo aver analizzato alcune caratteristiche di autonomia presente in natura, evidenziando come esse trovino concreta forma nel MAS rappresentato dal gioco Robot Wars, è interessante osservare come l'autonomia, così come formalizzata in **[Ros14]** emerga dal caso di studio.

L'autonomia dei sistemi consiste, secondo **[Ros14]** nella "*caratteristica di mantenere forma e funzione nel tempo e di acquisire una flessibilità auto determinata"*:

- **Rete interna di processi, che mantiene il sistema** —> Tale aspetto è assimilabile ai piani di cui gli agenti sono dotati, i quali in base al contesto e agli obiettivi e/o eventi permettono di mantenere gli agenti stessi, spingendoli a reagire ai pericoli, attaccando o fuggendo ad esempio.

- **Confini con l'ambiente e regolazione di scambi ed interazioni** —> Come già menzionato, il paradigma ad agenti prevede una separazione tra agente ed ambiente, e scambi di informazioni: in ingresso all'agente mediante i ***percepts*** e in uscita dall'agente per mezzo di **azioni** che hanno come oggetto l'ambiente stesso. Le azioni spaziali dei robot sono sempre legate, oltre che ad altri fattori interni, alla situazione del mondo.

- **Regole di comportamento auto specificate, per reagire agli stimoli in modi auto determinati, in base alla propria disposizione interza e condizione** —> In relazione ai *percepts* e a stimoli quali informazioni provenienti da altri agenti (***messaggi***) e alla propria conoscenza i robot si trovano ad attuare i piani che permettono di raggiungere il goal prefissato tenendo conto del completo contesto in cui si trovano.

- **Interdipendenza e Differenziazione** —> a livello di MAS, abbiamo una differenziazione degli agenti, i quali incarnano tipologie di robot differenti, e l'interdipendenza è esplicita quando le azioni di una categoria di robot è influenzata dalla presenza di altri robot (come il rapporto di rifornimento tra Dispenser ed Archer oppure i piani di inseguimento e fuga di cui sono dotati gli Archer, gli Attacker e il Player).

- **Autonomia Temporale** —> Come menzionato nel paragrafo precedente, osserviamo una certa ciclicità.

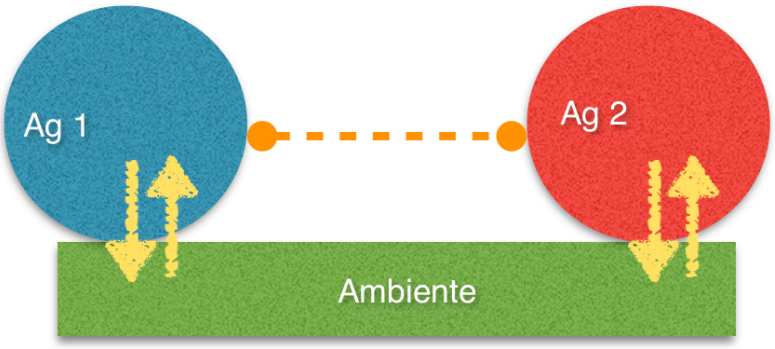
- **Robustezza** —> Già menzionata, ricordando in particolare la gestione di situazioni impreviste mediante un ricco spettro di possibili piani e una efficace gestione dei fallimenti, per poi citare la possibilità di aggiornare le credenze dei robot e le percezioni.

Modello Agent-Based

Il caso di studio utilizza **JASON** come incarnazione di **AgentSpeak**, un linguaggio astratto di descrizione e programmazione di agenti secondo il Framework BDI (Beliefs Desires Intentions). Prima di approfondire questi dettagli tecnologici è necessario osservare come il paradigma ad agenti sia effettivamente un paradigma per modellare entità e sistemi autonomi. Nelle parti precedenti sono state già mostrate alcune caratteristiche e proprietà che abbracciano l'autonomia, ma mettere in luce le peculiarità teoriche del paradigma permette di osservare a tutto tondo il corpus di tale framework concettuale.

Il primo aspetto promotore di autonomia è proprio a livello di paradigma stesso: gli agenti non hanno interfaccia e non possono essere controllati ed invocati. Inoltre gli agenti, come già evidenziato, incapsulano il proprio flusso di controllo e ciò è in conformità con il concetto di *boundaries*, permettendo solo lo scambio di informazioni con l'esterno.

Il legame con il **contesto** è estremamente forte, pertanto si parla di **entità situate**. L'importanza dell'ambiente non si esaurisce qui, anzi è possibile dire che l'ambiente gioca un ruolo di mediatore e dunque fornisce essenzialmente un **sistema di coordinazione** delle entità situate in esso **nature-based.**



(*La linea tratteggiata rappresenta la comunicazione indiretta ottenuta mediante la mutua interazione dei due agenti con il mondo*)

Questo tipo di comunicazione indiretta è dominante nel caso di studio proposto: la maggior parte delle scelte dei robot sono legate alla percezione che hanno riguardo alla loro posizione spaziale e alle entità circostanti, e le azioni quali lo spostamento sulla griglia di gioco sono interpretabili come

una notifica di mutamento ambientale ai robot circostanti. Nelle sezioni successive questo tema sarà ulteriormente approfondito.

Rimanendo sul tema della coordinazione, i robot sono stati dotati di ***social ability*** in quanto spesso devono interagire tra di loro in maniera diretta. Gli Archer notificano la necessità di rifornimento ai Dispenser, i quali collaborano rifornendoli, e (in riferimento alla versione estesa del gioco), i meccanismi di fusione costituiscono di fatto un'incarnazione di cooperazione mediante *social ability*. Infine, il rapporto tra il Player e i robot di offesa è una forma di coordinazione mediante competizione in quanto i colpi subiti, uniti al contesto ambientale, hanno come effetto quello di regolare i mutui spostamenti.

L'architettura ad agenti consente di realizzare proprietà di autonomia come come la reattività e la proattività:

Reattività—> secondo i principi di omeostasi, situatedness e robustezza, gli agenti sono in grado di rispondere agli stimoli in tempo utile.
Proattività —> gli agenti non solo sono in grado di reagire, ma anche di "agire prima", ossia portare avanti sotto procedure per avvicinarsi alla realizzazione di un goal/task.

Infine gli agenti possiedono uno **stato**, interpretabile come un corpus di conoscenze e percezioni, secondo le quali agire: i robot svolgono la maggior parte delle loro azioni tenendo conto di informazioni acquisite anche in istanti precedenti oltre che all'istante corrente.

Tipologie di Autonomia riscontrate

Riassumendo dunque le tipologie di autonomia riscontrate in questa prima analisi sono:

- **Autonomia Sociale** —> Autonomia in relazione agli altri agenti.
- **Autonomia Interattiva** —> Autonomia in relazione all'ambiente.

Stigmergia

Citando l'ambiente e la sua funzione di medium di coordinazione è interessante riportare che **[Gra59]** in relazione agli studi sulla costruzione dei termitai osservò che :

"il coordinamento delle attività e la regolamentazione delle costruzioni non sono direttamente dipendenti dai lavoratori, ma dalle costruzioni stesse ."

Le termiti, partendo da un insieme di regole decentralizzate, relativamente semplici, riescono a generare strutture complesse: ogni termite produce una sfera di fango e la ricopre di feromoni e le altre termiti, attratte dall'odore di individui dello stesso termitaio, sono spinte a depositare le loro sfere di fango accanto a quelle già depositate. In questo modo si formano strutture complesse come archi pilasti e gallerie.
Questo meccanismo di comunicazione decentralizzato, sfruttando la reciproca modificazione dell'ambiente, prende il nome di Stigmergia. Anche le formiche utilizzano una forma di stigmergia, in quanto, utilizzando i feromoni e modificando dunque l'ambiente, riescono a comunicare informazioni quali la direzione da percorrere.

In Robot Wars il comportamento stigmergico è osservabile in meccanismi quali le fasi di inseguimento. Un robot che insegue un nemico agisce sull'ambiente modificando la sua posizione.

in risposta il robot attaccato (che ipotizziamo sia in condizione di non poter contrattaccare, ad esempio per la poca vita rimasta o per una mancanza di munizioni) modifica la sua posizione in modo da allontanarsi in maniera efficace dalla posizione appena occupata dal suo nemico.

Agenti come sistemi intenzionali

L'**atteggiamento intenzionale** (***intentional stance***) è un'astrazione per capire, guidare e descrivere il comportamento dei sistemi.

"L'atteggiamento intenzionale è la strategia di interpretare il comportamento di un soggetto (persona, animale, artefatto, a prescindere) trattandolo come se fosse un agente razionale che ha governato la sua 'scelta' di 'azione' da un 'osservazione' delle sue 'credenze' e 'desideri' " **[Den07]**

Pertanto la scelta di modellare gli agenti come entità dotate di stati mentali è particolarmente efficace, in quanto è possibile rappresentare comportamenti complessi in maniera agevole e rappresenta un modo semplice di descrivere, comprendere e costruire agenti.

Nel caso di studio proposto i robot sono dotati delle due tipologie di stati mentali:

Stato epistemico —> **Conoscenza del mondo** : *Belief, Percept*. Abbiamo che i robot sono consapevoli, istante dopo istante, della loro posizione, delle entità nelle vicinanze e di alcune caratteristiche di esse, oltre a possedere delle credenze riguardanti la propria conformazione.

Stato motivazionale —> **Obiettivi dell'agente** : *Goal, Desire*. I robot in ogni istante hanno un obiettivo da raggiungere, che va dal movimento secondo lo schema di spostamento che li caratterizza, al calcolo di posizioni relative, alla fuga ecc…

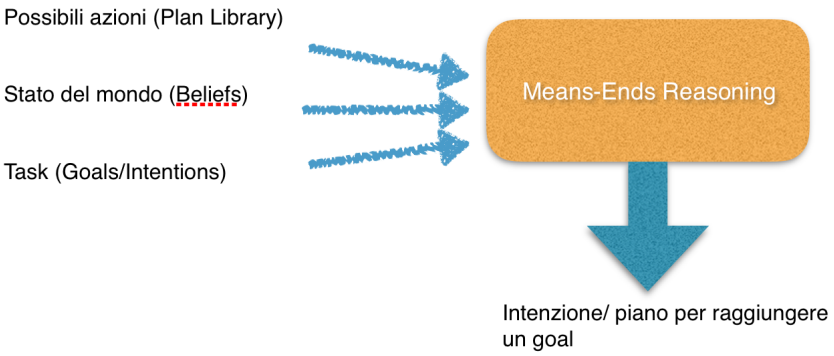
L'azione che viene eseguita tra quelle disponibili, basandosi sullo stato mentale attuale (quest'ultimo individuabile con il *context* dei pani) è detto **Ragionamento Pratico.**
Esso si forma di due categorie:

Deliberazione: è assimilabile al concetto di Intenzione, decisione di ciò che l'agente vuole raggiungere.
Ragionamento "means-ends": si focalizza su come ottenere ciò che si vuole raggiungere.

I robot sono fortemente caratterizzati in questo senso: il contesto di gioco deve prevedere una molteplicità piuttosto corposa di scenari possibili, e numerose variabili entrano in gioco di volta in volta. Rappresentando tutti i possibili stati mentali otterremmo un diagramma in cui l'esplosione degli stati è notevole. Basti pensare che la maggior parte delle azioni deve tener conto della vita del robot, dei robot nelle vicinanze, della loro tipologia e della conformazione dell'arena nelle vicinanze del soggetto.
Per questo motivo è presente una vasta scelta di scenari (o più propriamente, contesti) per ogni goal, proprio per permettere di scegliere l'insieme di procedure più indicato di volta in volta.

Un'altra tipologia di ragionamento interessante è il **Ragionamento epistemico**: esso è il processo con cui vengono aggiornate le informazioni di un agente in modo da mantenere sempre le credenze consistenti con lo stato attuale del mondo. I piani di movimento, fuga, avvicinamento all'interno di Robot Wars seguono questo tipo di ragionamento in quanto, una volta intrapresi, si occupano di (ri)generare conoscenza, in particolare riguardo all'apprendimento della prossima mossa da fare.

(Schema generale di sintesi del processo di ragionamento)



(Focus sul Means-Ends Reasoning)

Osservando l'*agent control loop* di un agente che voglia implementare il ragionamento pratico, troviamo una forte aderenza nella struttura dei robot presenti in robot wars:

1: while true

2: observe the world; **[aggiornamento percept e messaggi dagli altri robot]**

3: update internal world model (beliefs); **[aggiornamento beliefs sulla base del contenuto dei messaggi e dei percept ricevuti]**

4: deliberate which intention to adopt next; **[scelta tra i desire disponibili]**

5: use means-ends reasoning to get a plan for the given intention; **[individuato il goal da perseguire, adozione del piano adeguato al contesto]**

6: execute the plan; **[esecuzione dei passi del body del plan, aggiornamento eventuale di belief e desire]**

7: end while;

Inquadriamo i robot nello schema **Beliefs, Intentions, Desires**.

B—> Conoscenza riguardo le proprie caratteristiche (vita, potenza, latenza, colpi disponibili, e prossimo spostamento da attuare, …), le percezioni ambientali (posizione attuale, robot nelle vicinanze, muri, …) e le informazioni ricevute dagli altri agenti (hit, dead, …)

D—> Goal che il robot vuole raggiungere; tranne il caso del Dispenser il cui goal principale è il monitoraggio continuo dell'ambiente, gli altri robot hanno come goal principale quello di muoversi costantemente. Altri goal come il calcolo della direzione rispetto ad una entità, la fuga o l'attacco possono essere sotto goal dei piani del robot.

I —> Piani ("ricette" procedurali per raggiungimento goal) e post condizioni. Ampia varietà di contesti, al fine di gestire la ricchezza delle situazioni emergenti nel gioco.

AgentSpeak fornisce a livello di linguaggio tre costrutti principali, *Belief* (lo stato dell'agente e info ambientali), *Goal* (lo stato che l'agente vuole raggiungere) e *Plan* (ricette procedurali per il conseguimento dei goal). L'architettura di un agente in AgentSpeak possiede quattro costrutti principali:
Base di credenze (*belief base*), Libreria di Piani (*Plan Library,*) Insieme di eventi (*set of events*) e Insieme di intenzioni (*set of intentions*).

L'incarnazione di AgentSpeak utilizzata nel progetto è **JASON**, un framework scritto in Java che implementa la semantica operazione di una variante di AgentSpeak e fornisce nuovi meccanismi come la definizione di nuove azioni interne, la gestione dei fallimenti, le annotazioni, lo scambio di messaggi e la possibilità di avere un environment simulato.

Gerarchizzazione nel Means-End Reasoning

Una volta individuato il goal da conseguire, la scelta dei piani per tale goal è eseguita consultandoli in maniera sequenziale fino ad a raggiungere un contesto che unifica con la conoscenza attuale.
I piani dei robot per una stessa *wanderMove* sono inseriti in ordine tale da avere come piano base lo spostamento generico. Se dunque il robot si trova ad eseguire un piano collocato inferiormente, significa che gli input che ha ottenuto o le conoscenze che ha ricavato non hanno unificato con il piano precedente. Certi piani inoltre possono essere dotati di un *context* più snello in quanto certe informazioni possono essere date per consolidate o assenti per il semplice fatto che sarebbero state unificate in precedenza. Riportiamo una piccola parte dell'Archer robot come esempio:

```
+!wanderMove(d,A) :  nearMe(player_robot,N,_,_,L,_)
                                & L>1
                                & attack_power(F)
                                & dilitium_bullets(B)
                                & B>0



+!wanderMove(d,A) : nearMe(player_robot,N,_,_,L,_)
                                & L<=1
                                & attack_power(F)
```

```
                                & life(V)
                                & V> 20
                                & dilitium_bullets(B)
                                & B>0


+!wanderMove(d,A) :  nearMe(dispenser_robot,_,_,_,L,X) &
dilitium_bullets(B) & B<4 & not dead(X)
                                & L>1
                                & not nope(X)


+!wanderMove(d,A) :  nearMe(dispenser_robot,_,_,_,L,X) &
dilitium_bullets(B) & B<4  & not dead(X)
                                & L<=1
                                & not nope(X)


+!wanderMove(d,A) : nearMe(player_robot,_,_,_,_,_)
                                 & dilitium_bullets(B)
                                 & life(L)
                                 & (B==0 | L<=20)
```

Nel codice abbiamo tralasciato i piani collocati superiormente quindi ci concentriamo sulla gerarchia relativa solo a questi contesti. I primi due piani hanno come corpo del piano (non riportato per motivi di spazio) la procedura per attaccare il Player. L'ultimo piano invece è un piano di fuga dal Player. Nel mezzo ci sono due piani che consentono all'arciere, una volta individuato il Dispenser, di ricaricare i propri proiettili di dilitio. mediante la gerarchizzazione dei piani si inferisce che, rispetto alla fuga, è prioritario il rifornimento mentre è prioritario l'attacco rispetto al rifornimento. Si noti che non si è rivelato necessario specificare, oltre alla presenza dei Dispenser nel terzo e quarto piano, l'assenza del Player: se esso è presente e posso, lo attacco, se arrivo ad eseguire il rifornimento è evidente che il ***means-ends reasoning*** non ha unificato con la possibilità di attaccare.

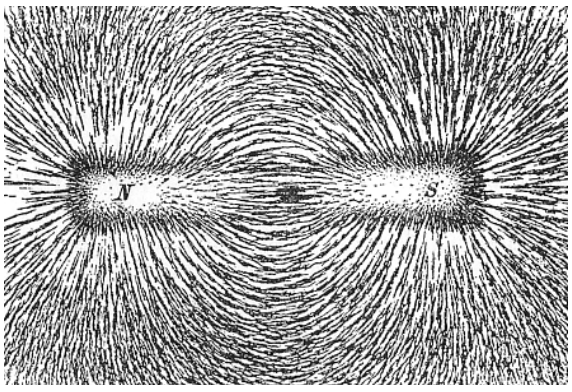
Estensione: auto organizzazione ispirata alla natura

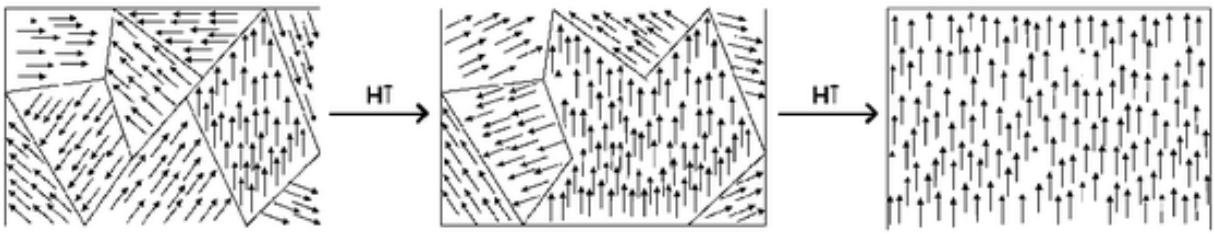
La versione estesa del progetto introduce il tentativo di realizzare una forma di auto organizzazione di ispirazione naturale, attraverso tecniche basilari.
Tale versione si può definire in stato di “Alpha preview” in quanto non sono state implementate tutte le ottimizzazioni di gestione della fusione, ma offre alcuni spunti interessanti in fatto di Self Organization. Inoltre questo approccio ha permesso di individuare come la difficoltà di gestione di una coordinazione così complessa necessiti da parte dello sviluppatore uno sforzo rappresentativo notevole e giustifica ampiamente la necessità di introdurre astrazioni più espressive in ambito di coordinazione orientata alla auto organizzazione.

Sostanzialmente si vuole fare in modo che gli Archer e gli Attacker siano dotati della possibilità di unire le forze, incrementando le loro caratteristiche attraverso una sorta di simbiosi.
Per evitare che uno dei due elementi della fusione perdesse in autonomia diventando passivo nei confronti dell’altro si è scelto di non utilizzare lo scambio di messaggi al fine di fornire nuovi piani o goal, bensì di inserire un corpus di piani identico nei due agenti in modo da attuare lo stesso spettro di piani nel caso il goal sia quello di operare durante una fusione in corso: ammettendo che la sincronizzazione possa avviarsi indistintamente da un Attacker o da un Archer, sarebbe comunque stata necessaria una differenziazione e duplicazione di piani, perdendo dunque in sinteticità.

Si è scelta questa strategia anche al fine di rispettare il principio dell’***autonomia motivazionale***, la quale prevede che un entità non possa imporre goal ad un’altra, ma solo le sue percezioni tramite interazione(**[Cas95]**).

L’idea che ha condotto questa sperimentazione parte dal principio fisico della **polarizzazione magnetica** come metafora da cui trarre ispirazione: questo fenomeno produce in alcuni metalli l’allineamento dei propri atomi in modo da orientarsi in direzione parallela al campo magnetico a cui sono sottoposti:





Citando un altra similitudine con la natura, le celle di Bénard, che descrivono il moto di un fluido per effetto convettivo quando gli viene fornito calore dal basso, sono un altra valida fonte di ispirazione.

Prendendo infine a campione il mondo animale, un esempio di auto organizzazione rilevante è quello dei banchi di pesce; in essi non c'è un individuo dominante che impone il movimento agli altri, bensì viene utilizzato un **set di regole locali comuni a ciascun elemento del banco**:
1)Mantenere una certa distanza dal pesce vicino
2)Continuare a nuotare
Partendo quindi da un comportamento comune e semplice a livello locale, si ha un fenomeno emergente assai più complesso. **[Jac]**

Si vuole realizzare una forma di autorganizzazione analoga, in cui le singole entità possiedono **intrinsecamente** la capacità di adattarsi. Per questo motivo ogni entità che fa parte della fusione sa spontaneamente come deve comportarsi: in questo modo si genera il fenomeno emergente del moto comune dei due partecipanti alla fusione.
Di seguito sono riportate alcune schermate che mostrano l'applicazione di tale processo:

```
[attacker_robot3] ()()()()() ho visto il mio AMICO!!: archer_robot6
[attacker_robot8] ATT-WANDER!--> TROVATO IL PLAYER; essendo a distanza 1 e avendo ancora 35 di vita, COLPISCO!
[player_robot] colpito
[attacker_robot8] seguo
[player_robot] PLAYER ROBOT --> TROVATO QUALCUNO archer_robot6, MOLTA VITA ma è lontano MI AVVICINO
[player_robot] seguo
[archer_robot6] ricevuta unione daattacker_robot3che è nel punto giusto
[archer_robot6] --STO FERMO--
[attacker_robot3] ORA POSSO SINCRONIZARe
[attacker_robot3] /=======================sync
[archer_robot6] UPDATE of my PARAMS
[archer_robot6] 00000000000000000000
[attacker_robot3] 00000000000000000000
[archer_robot6] utilizzo abilità di attaccante; COLPISCO e sto lì
[attacker_robot3] utilizzo abilità di attaccante; COLPISCO e sto lì
```

(Dopo la fusione si sviluppa una forma di auto organizzazione per cui localmente ogni membro della fusione sa come deve comportarsi, e collettivamente si assiste al fenomeno emergente che corrisponde alla fusione)

Agli occhi del Player non è percepita tale collettività motivo per cui esso colpisce soltanto uno dei due robot rilevati. Per fare ciò è stata aggiunta una politica di interazione mediante messaggio alla fine di aggiornare le percezioni di entrambi i membri di questa simbiosi:

```
[attacker_robot3] utilizzo abilità di attaccante; COLPISCO e sto lì
[archer_robot6] utilizzo abilità di attaccante; COLPISCO e sto lì
[player_robot] colpito
[player_robot] colpito
[player_robot] PLAYER ROBOT --> TROVATO QUALCUNO,archer_robot6 ho molta vita ed è VICINO!  SPADATA !
[player_robot] FENDENTE della SPADA !
[archer_robot6] fusion: colpito!!
[attacker_robot3] fusion: colpito di rimando!!
[archer_robot6] utilizzo abilità di attaccante; COLPISCO e sto lì
[attacker_robot3] utilizzo abilità di attaccante; COLPISCO e sto lì
[player_robot] colpito
[player_robot] colpito
[player_robot] PLAYER ROBOT --> TROVATO QUALCUNO,archer_robot6 ho molta vita ed è VICINO!  SPADATA !
[player_robot] FENDENTE della SPADA !
[archer_robot6] fusion: colpito!!
[attacker_robot3] fusion: colpito di rimando!!
[player_robot] PLAYER ROBOT --> TROVATO QUALCUNO,archer_robot6 ho molta vita ed è VICINO!  SPADATA !
[player_robot] FENDENTE della SPADA !
[archer_robot6] fusion: colpito!!
[attacker_robot3] fusion: colpito di rimando!!
```

L’ultima immagine proposta mostra invece un esempio di rifornimento durante la fusione:

```
[attacker_robot4] FUSION-----TROVATO IL DISPENSER dispenser_robot2 ; essendo a distanza 3 e avendo rimasto 0 colpi rimasti, MI AVVICINO!
[archer_robot6] FUSION-----TROVATO IL DISPENSER dispenser_robot2 ; essendo a distanza 3 e avendo rimasto 0 colpi rimasti, MI AVVICINO!
[attacker_robot4] FUSIONseguo
[archer_robot6] FUSIONseguo
[attacker_robot5] ***LIBero di muovermi, niente ostacoli
[archer_robot1] ***LIBero di muovermi, niente ostacoli
[attacker_robot4] FUSION-----TROVATO IL DISPENSER dispenser_robot2 ; essendo a distanza 2 e avendo rimasto 0 colpi rimasti, MI AVVICINO!
[archer_robot6] FUSION-----TROVATO IL DISPENSER dispenser_robot2 ; essendo a distanza 2 e avendo rimasto 0 colpi rimasti, MI AVVICINO!
[attacker_robot4] FUSIONseguo
[archer_robot6] FUSIONseguo
[attacker_robot5] ***LIBero di muovermi, niente ostacoli
[archer_robot1] ***LIBero di muovermi, niente ostacoli
[archer_robot6] FUSION-----TROVATO IL DISPENSER dispenser_robot2 ; essendo a distanza 1 e avendo rimasto 0 colpi rimasti, RIFORNIMENTO!
[attacker_robot4] FUSION-----TROVATO IL DISPENSER dispenser_robot2 ; essendo a distanza 1 e avendo rimasto 0 colpi rimasti, RIFORNIMENTO!
[dispenser_robot2] %%%%%%%%%% rifornimento
[dispenser_robot2] %%%%%%%%%% rifornimento
[archer_robot6] FuSIONfuga con i valori di D=d; V=1
[attacker_robot4] FuSIONfuga con i valori di D=d; V=1
```

Problemi riscontrati

Dal momento che questa estensione del progetto di base è stata condotta perlopiù a scopo sperimentale, manca nell’architettura una completa gestione di scenari critici, pena una cattiva robustezza degli agenti alla sincronizzazione; tuttavia essi continuano ad essere robusti nel loro singolo comportamento.

Sviluppi futuri

Tra le possibili migliorie ed estensioni per Robot Wars vi è la creazione di nuove tipologie di robot, al fine di rendere più vario lo scenario di gioco. I robot previsti sono l’**Healer Robot**, un robot in grado di curare gli altri robot nemici e il **Defender Robot**, un robot in grado di muoversi, ma incapace di attaccare, caratterizzato da una grande longevità.
Sono inoltre ipotizzabili ulteriori tipi di fusione, possibilmente caratterizzati da altre meccaniche: ne è un esempio una possibile unione tra Attacker e Defender, in cui quest’ultimo si sacrifica per conferire al destinatario la sua longevità.

Bibliografia

[Tho10] Evan Thompson.
Mind in Life: Biology, Phenomenology, and the Sciences of Mind. Belknap Press, 2010.

[Kit02] Hiroaki Kitano.
Foundations of Systems Biology. MIT Press, 2002 .

[Ros14] Bernd Rosslenbroich.
On the Origin of Autonomy. A New Look at the Major Transitions in Evolution.
History, Philosophy and Theory of the Life Sciences. Springer International Publishing, 2014.

[Den07] Daniel Dennett.
Intentional systems theory.
In Ansgar Beckermann and Sven Walter, editors, Oxford Handbook of the Philosophy of Mind, chapter 19. Oxford University Press, 2007.

[Gra59] Pierre-Paul Grassé.
La reconstruction du nid et les coordinations interindividuelles chez Bellicositermes natalensis et Cubitermes sp. la théorie de la stigmergie: Essai d'interprétation du comportement des termites constructeurs. Insectes Sociaux, 6(1):41–80, March 1959.

[Cas95]Cristiano Castelfranchi.
Guarantees for autonomy in cognitive agent architecture.
In Michael J. Wooldridge and Nicholas R. Jennings, editors, Intelligent Agents, volume 890 of Lecture Notes in Computer Science, pages 56–70. Springer Berlin Heidelberg, 1995.
ECAI-94 Workshop on Agent Theories, Architectures, and Languages (ATAL) Amsterdam, The Netherlands 8–9 August 1994. Proceedings.

[Jac]Nathan S. Jacob.
How do schools of fish swim in harmony?
Ted Ed.