Scontro di squadre di robot in Robocode

Francesco Baldisserri

4th January 2006

Abstract

Un'aspetto interessante della modellazione tramite sistemi multi-agente risiede nella possibilità di sviluppare semplici elementi software orientati allo svolgimento di task elementari e riuscire, nell'insieme, a risolvere problemi complessi, grazie al comportamento emergente di questa popolazione.

Scopo di questo progetto è la realizzazione di squadre di robot software che si scontrano in un'arena utilizzando diverse strategie di collaborazione. Queste prove sono eseguite in un framework ad-hoc, Robocode [2], sviluppato in Java, in cui è possibile sviluppare i robot e farli combattere tra loro singolarmente o in team.

1 Obiettivo

Come anticipiato nell'abstract l'obiettivo primario di questo progetto è la realizzazione ed il confronto di due squadre di robot:

- La prima squadra si basa su una collaborazione implicita, ovvero ogni robot fa del suo meglio per aiutare i compagni e sconfiggere gli avversari, ma le uniche informazioni a disposizione del singolo elemento sono quelle percepite (e quindi niente di più di quello che vedono/sentono anche gli avversari).
- Nella seconda squadra si realizza invece una collaborazione esplicita, i robot si scambiano messaggi, mettendo al corrente i compagni delle loro intenzioni o richieste, permettendo in questo modo una pianificazione strategica d'insieme.

Da questi esperimenti si vuole capire se il comportamento collaborativo basato su comunicazione è superiore al comportamento che semplicemente emerge dalla combinazione dei singoli comportamenti dei membri del team dove non avviene scambio esplicito d'informazione.

Essendo il progetto focalizzato in particolare sul lavoro di squadra, i singoli bot non sono particolarmente sviluppati; considerato che realizzare anche solo un singolo bot vagamente intelligente occuperebbe ben più di un corso, ci si limita a bot dal comportamento coerente ma semplice, il più possibile uguale per tutte le squadre, in modo da poter valutare il vantaggio evantualmente tratto da una certa tipologia di collaborazione.

2 Analisi

2.1 Perchè gli agenti?

In primo luogo è necessario valutare se i robot, mattoni fondamentali del sistema, possono essere descritti come agenti e, soprattutto, se questo approccio può semplificare lo sviluppo progetto; per fare questo si considerano alcune definizioni e proprietà che comunemente si associano a questa astrazione.

> "An agent is a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to meet its design objectives". [3]

I concetti che si possono individuare in questa definizione sono:

- computer system: soddisfatto, essendo il robot che si intende sviluppare un componente software.
- situated in some enviroment: soddisfatto poichè i robot sono immersi in un ambiente in cui possono agire e riceve stimoli.
- capable of autonomous action: soddisfatto, in questo framework (Robocode [2]) i robot sono indipendenti gli uni dagli altri ed ognuno decide autonomamente le proprie azioni.
- in order to meet its design objectives: soddisfatto, essendo un gioco, l'obiettivo di ogni agente è vincere, collaborando coi compagni nel caso di partite a squadre.

I robot aderiscono quindi alla definazione sopra citata ed inoltre soddisfano anche quella fornita durante il corso (MAS Agent):

> "Entità autonoma situata che persegue il suo goal interagendo con l'esterno.
> -autonomia & proattività
> -interattività (reattività , situatedness)"

P.S: L'agente in questione non soddisfa la definizione classica dell'AI cioè non è intelligente; i robot realizzati saranno, per forza di cose, semplici, reagenti ad eventi esterni e con una (piccola) memoria riguardo al passato, senza però che si includano caratteristiche come flessibilità, learning...

A valle di quanto detto in questo paragrafo appare ragionevole utilizzare gli agenti per questo particolare progetto. Si fa notare come il livello d'astrazione che gli agenti introducono faciliti la descrizione di un robot (o una squadra di essi), sarebbe infatti difficile realizzare lo stesso sistema senza ricorrere a concetti come proattività, goal, autonomia, enviroment...

2.2 Cos'è Robocode?

Come già detto robocode è un framework java che fornisce il supporto per la realizzazione di robot per farli poi scontrare in un'arena virtuale. Il sistema è organizzato a turni e segue questo flusso:

per ogni ciclo:

disegna i robot, i proiettili e le esplosioni
per ogni robot:

sveglia il thread del robot corrente
aspetta che il robot chiami un funzione bloccante,
fino ad un massimo intervallo di tempo

fine per ogni
svuota le code degli eventi dei robot
sposta i proiettili, ed eventualmente genera degli eventi nelle code dei robot
sposta i robot, ed eventualmente genera gli eventi nelle code dei robot
finalizza il campo di battaglia e genera eventualmente altri eventi nelle code dei robot
attendi per adeguarti al frame-rate impostato

fine per ogni

Ogni ciclo è detto tick del sistema e ad ogni tick corrisponde un frame nella visualizzazione a schermo dell'azione. Ad ogni turno, ogni robot dispone di un certo tempo entro il quale deve eseguire delle azioni; le azioni a disposizione di ogni robot riguardano:

- Radar: è l'occhio del robot e permette di ricavare informazioni riguardo all'ambiente circostante, può essere ruotato a piacere ed ogni volta è puntato su un tank viene generato un evento contente diverse informazioni relative al soggetto in questione.
- Movimento: ogni robot può muoversi liberamente in uno spazio dimensionale delimitato dai muri dell'arena; quando il bot colpisce uno di questi muri o un altro bot, oltre a ricevere un danno, viene generato un evento che riporta tipo di ostacolo, direzione etc...
- Cannone: è la principale arma offensiva a disposizione (alcuni tank fanno anche del ramming, ovvero caricano gli avversari), può ruotare indipendentemente dal resto e per ogni colpo sparato si genera un evento di "hit" o "missed" a seconda che il colpo abbia avuto successo o meno.

Quella sopra esposta è solo un'idea di come sia strutturato il sistema, si sono, per forza di cose, trascurati molti dettagli ed eventi presenti nel framework (per ulteriori dettagli si veda [2]).

Riguardo al sistema di comunicazione tra bot non ci sono scelte obbligate ma, considerando che le esigenze sono basilari si vogliono privilegiare i meccanismi presenti nell'ambiente di simulazione, evitando di appoggiarsi ad un'architettura esterna. La collaborazione tra i bot sarà quindi basata su uno scambio di messaggi, unica forma di comunicazione supportata. Questo metodo è meno restrittivo di quanto possa sembrare poichè i messaggi sono generici e possono portare praticamente qualsiasi cosa (sono oggetti Java) e quindi, in questo caso, anche artefatti per la collaborazione.

3 Progetto

Il progetto del robot si articola in due fasi principali, lo sviluppo del singolo tank e la definizione delle modalità di collaborazione. L'ottica di sviluppo è orientata ad una definizione semplice e chiara del comportamento, si lascia poco o nessun margine ad oscuri metodi di AI per ottenere comportamenti complessi ma si preferisce cablare a priori le pratiche considerate migliori ("a little intelligence goes a long way!").

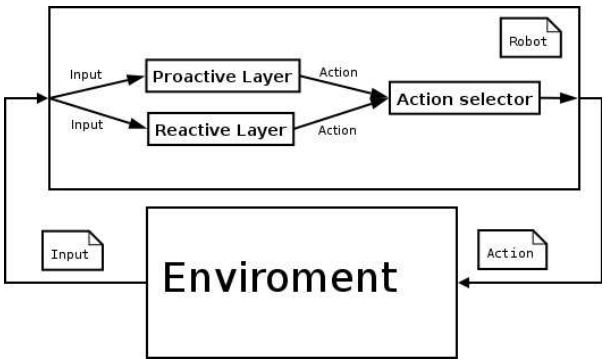
3.1 Architettura del robot

L'architettura adottata è di tipo ibrido e combina i comportamenti di due livelli, uno reattivo ed uno proattivo (per una trattazione più dettagliata riguardo ai sistemi reattivi ed ibridi si veda il capitolo 5 di [3]). Una semplice rappresentazione di com'è composto il robot (e di come interagisce con l'ambiente) è data in figura 1.

Gli input forniti dall'ambiente sono percepiti da entrambi i layer ed ognuno di essi può elaborare, indipendentemente dall'altro, un'azione da suggerire al selettore delle azioni; il selettore delle azioni (action selector in figura 1) decide per una delle possibili alternative suggerite dai layer. Si fornisce a questo punto una descrizione dei singoli elementi presenti all'interno del robot.

Action selector Si inizia da questo componente perchè condiziona fortemente il comportamento del bot; in questo caso particolare si è optato per una politica di selezione banale, si privilegiano sempre le azioni "reattive" a discapito di quelle "proattive". Questa scelta deriva direttamente dalla subsumption architecture proposta da Brooks (rif. [1]), ed è molto semplice e ragionevole (se metto una mano sul fuoco la tolgo e basta, indipendentemente dal fatto che voglia andare a giocare a basket nel pomeriggio!). In questo scenario le reazioni diventano veri e proprio istinti che il tank soddisfa indipendentemente dal proprio obiettivo; una conseguenza immediata di questa struttura è evidente: un determinato insieme di reazioni preclude un altro insieme di goal che, a questo punto, non sono più perseguibili.

Figure 1: Struttura ibrida del robot



P.S. Da quanto appena detto potrebbe sembrare che in presenza di azioni reattive si scartino in ogni caso quelle proattive, in realtà, se queste non accedono alle stesse risorse possono tranquillamente convivere (es: ruotare il radar mentre ci si allontana dal muro).

Reactive layer La parte reattiva si occupa della gestione di tutti quei problemi di basso livello che si vuole rendere trasparenti alla parte proattiva. In particolare:

- Se urti o sei troppo vicino ad un muro -> allontanati dal muro
- Se urti un altro robot -> allontanati dal robot
- Se vieni colpito da un proiettile -> spostati

L'attuazione di strategie offensive/difensive è lasciata al livello superiore, come già detto, lo scopo di questa parte è rendere trasparenti quei dettagli che non si vuole vengano considerati altrove.

Proactive layer Questo livello è decisamente più sofisticato del precedente, a questo proposito lo si è scomposto in componenti fondamentali, ognuno dei quali gestisce un'aspetto indipendente del robot ovvero radar, movimento e cannone (vedi figura 2).

RadarModule Il radar è l'occhio del robot ed ha lo scopo di localizzare i robot e ricavarne informazioni. In particolare questo gestore ha due modi di funzionamento principali:

- Scanning: ruota il radar continuamente su tutto il campo permettendo di visualizzare (uno alla volta) tutti i robot presenti, utile per mantenere una visione d'insieme dello stato corrente del gioco.

- Tracking: mantiene il radar su un robot, perdendo di vista in questo modo il resto del mondo, ma disponendo sempre di informazioni (posizione, energia, velocità, etc...) aggiornate del target.

VehicleModule Si occupa di muovere il bot secondo la modalità scelta, le modalità di movimento disponibili sono:

- Random: il robot si muove per lo schermo seguendo uno schema casuale, utile quando non si ha un obiettivo particolare poichè è più difficile essere colpiti.
- Follow: il robot si mantiene entro un certo range di distanza dal target da seguire (amico o nemico che sia), utile per colpire un nemico in particolare o correre in soccorso di un compagno di squadra.

P.S: Si ricorda che gli urti contro altri robot o muri sono gestiti in maniera indipendente dalla modalità di movimento (sono infatti gestiti al livello inferiore).

GunModule Questo componente ha lo scopo di danneggiare gli avversari tramite l'utilizzo del cannone (gun). In questo gestore sono infatti contenute le procedure che valutano quando è conveniente o meno sparare e qual'è il modo migliore per farlo. Anche questo elemento ha due modalità fondamentali di funzionamento:

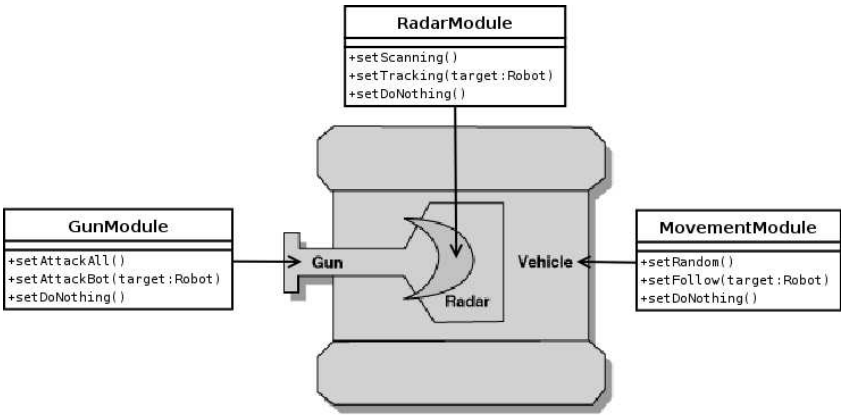
- Attack All: spara indistintamente verso tutti gli avversari che vede (tramite il radar).
- Attack Bot: spara solamente verso il target designato.

N.B. Comune a tutti i moduli esiste uno stato "Do nothing" dove il funzionamento del modulo può essere temporaneamente interdetto.

ModuleBot Questa sezione descrive l'entità robot vera e propria, i comportamenti desiderati sono ottenuti combinando le modalità di funzionamento dei moduli delle componenti del tank (i comportamenti reattivi rimangono sempre uguali). Si sono individuati tre modi di funzionamento (o, anche se improprio, stati) fondamentali:

1. Attack All: in questo stato il tank attacca tutti i nemici che vede muovendosi a random nell'area di gioco.
   (RadarModule: Scanning, VehicleModule: Random, GunModule: Attack All)
2. Attack Bot: il robot segue un nemico e lo attacca continuamente.
   (RadarModule: Tracking, VehicleMovementModule: Follow, GunModule: Attack Bot)

Figure 2: Moduli relativi alle componenti del tank



3. Defend Bot: in questa modalità si segue un compagno di squadra, attaccando tutti i nemici rilevati.
   (RadarModule: Scanning, VehicleModule: Follow, GunModule: Attack All)

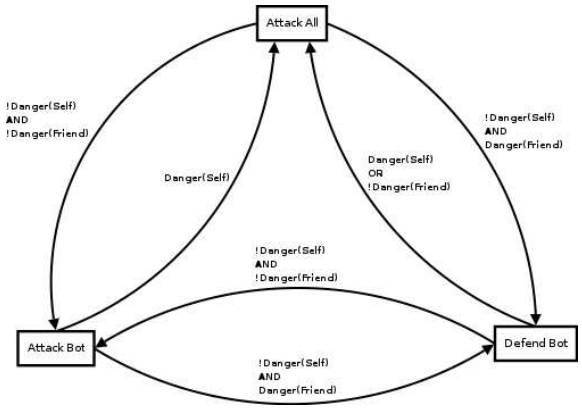
Osservando quanto appena descritto il bot potrebbe sembrare fin troppo "dumb", in realtà si devono considerare questi stati come macro-azioni a disposizione dell'agente, tramite le quali perseguire una strategia di gioco. Appare quindi evidente come sia fondamentale la scelta della modalità corrente di funzionamento considerando il proprio stato e quello del resto del mondo, sarà proprio la politica che guida le transizioni tra queste modalità a distinguere i robot che comporrano le due squadre.

Stati e transizioni Si può descrivere a questo punto lo schema per le transizioni tra gli stati del proactive layer. Come si vede in figura 3 la logica delle transizioni è molto semplice e si può descrivere in poche parole:

- se sei in pericolo pensa a te stesso (Attack All, la modalità più conservativa disponibile)
- se non sei in pericolo ed un tuo compagno lo è aiutalo (Defend Bot)
- se ne tu ne un tuo compagno siete in pericolo attacca un nemico (Attack Bot)

Si può notare come il concetto di "pericolo" sia assolutamente arbitrario, proprio in questa apparente ambiguità si differenziano le due squadre di tank che si vogliono sviluppare. In caso di assenza di comunicazione ogni robot deve

Figure 3: Diagramma degli stati



cercare di capire quando un compagno ha o meno bisogno di aiuto, quando invece è possibile la comunicazione ogni robot può chiedere rinforzi solo quando effettivamente necessario, eliminando quindi eventuali errori di valutazione o inefficienze.

3.2 Strategia e collaborazione

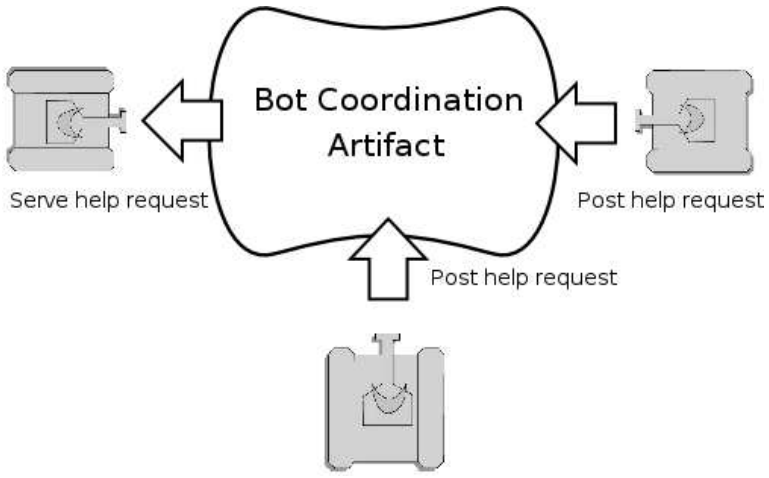
A questo punto del progetto le strategie delle due squadre sono praticamente definite, si tratta solamente di definire la frase "compagno in pericolo" nei due scenari distinti:

- In assenza comunicazione si ritiene che un compagno sia in pericolo quando il suo livello di energia scende sotto una certa soglia.
- In presenza di comunicazione un compagno è in pericolo se e solo se richiede espressamente aiuto.

Nell'ultimo caso si nota una maggiore flessibilità, non appena un compagno è in pericolo il robot ne viene a conoscenza immediatamente, mentre nel primo caso invece per rilevare un caso di pericolo era necessario scansionare col radar il compagno. In più la comunicazione può permettere una coordinazione tra i tank, senza che magari tutti i compagni corrano in soccorso di un altro ma dividendosi i compiti.

A valle di queste considerazioni si decide di implementare, come sistema di comunicazione, un artefatto dove ogni robot può depositare una richiesta di aiuto che sarà poi (forse) "servita" da un qualche compagno (figura). In questo prototipo l'artefatto è utilizzato solo come deposito per le richieste di aiuto, ma nulla vieta, in una sua estensione, di sofisticarne il comportamento aggiungendo la possibilità di impostare strategie o distribuire ordini.

Figure 4: Artefatto per la coordinazione tra robot



4 Implementazione

4.1 Barriere strutturali

La realizzazione del sistema progettato ha incontrato diverse difficoltà che ne hanno impedito una implementazione "elegante" e, in alcuni casi, coerente a quanto sopra descritto.

Il primo dei problemi incontrati riguarda l'impossibilità di sviluppare in maniera intuitiva una subsumption architecture, questo è dovuto al fatto che una modellazione/gestione consistente delle azioni dei robot è ostacolata dalla scarsa trasparenza degli effetti di un'azione stessa. In parole povere non è banale incapsulare (o meglio, atomizzare) un'azione, capire quando è effettivamente terminata e se va o meno in conflitto con altre azioni concorrenti. Tutti questi inconvenienti si sono riflessi nell'implementazione, che ha dovuto sopperire cablando l'architettura all'interno del tank; in pratica il comportamento è lo stesso di quello sopra definito, tuttavia il ruolo di layer, action selector ed actions è coperto dal bot che simula banalmente l'architettura tramite una struttura if/else.

Un'altra difficoltà riguarda il meccanismo di comunicazione, nonostante i robot possano scambiarsi qualsiasi cosa (Object Java) tramite messaggi, non è possibile mantenere un riferimento ad un oggetto comune a tutti i membri della squadra, di conseguenza non è stato possibile utilizzare l'artefatto sviluppato (BotCoordinationArtifact) come lavagna per la comunicazione all'interno del team. A questa mancanza si è dovuto sopperire con una comunicazione p2p tra i vari tank, dove ogni membro del gruppo richiede banalmente aiuto ai compagni tramite broadcast.

Table 1: Risultati

Membri	Vittorie Mute Team	Vittorie Talkative Team	Pareggi
1	511	474	15
3	477	514	9
5	485	513	2
7	469	528	3

4.2 Realizzazione

Nelle figure 5, 6 e 7 si riportano per completezza i diagrammi UML più significativi del progetto.

5 Risultati

Per valutare il comportamento delle due squadre si sono effettuati 1000 scontri tra team per diverse dimensioni (da 1 a 7 membri). Nella tabella 1 sono riportati i risultati.

Come si vede dai risultati il team basato su una collaborazione esplicita (Talkative team) ha un leggero vantaggio rispetto all'altro (Mute team), ma troppo piccolo per essere significativo; in parte questo è sicuramente dovuto ai problemi incontrati nell'implementazione tuttavia sono emerse anche ragioni più profonde che verranno esposte nelle conclusioni.

6 Conclusioni

Pattern Come già detto in sede d'analisi gli agenti realizzati non sono propriamente intelligenti, pur essendo presenti in rete diversi esempi di tecniche di intelligenza artificiale (reti neurali, algoritmi genetici, etc...) applicata a Robocode si è preferita un'intelligenza a design time. Esempi di questo tipo sono:

- predictive targeting tramite tecniche trigonometriche e/o d'estrapolazione
- tecniche di movimento casuale per il bullet dodging

Queste tecniche sono piuttosto semplici da realizzare e presentano comunque un buon comportamento; si può affermare che è generalmente preferibile affidarsi a tecniche classiche (ne è un esempio l'estrapolazione di funzioni) quando esse sono presenti invece che a tecniche di AI, meno efficienti e più difficili da realizzare.

Coordinazione I risultati riportati nel capitolo precedente evidenziano un aspetto importante della modellazione ad agenti, la coordinazione di diverse

Figure 5: Bots package

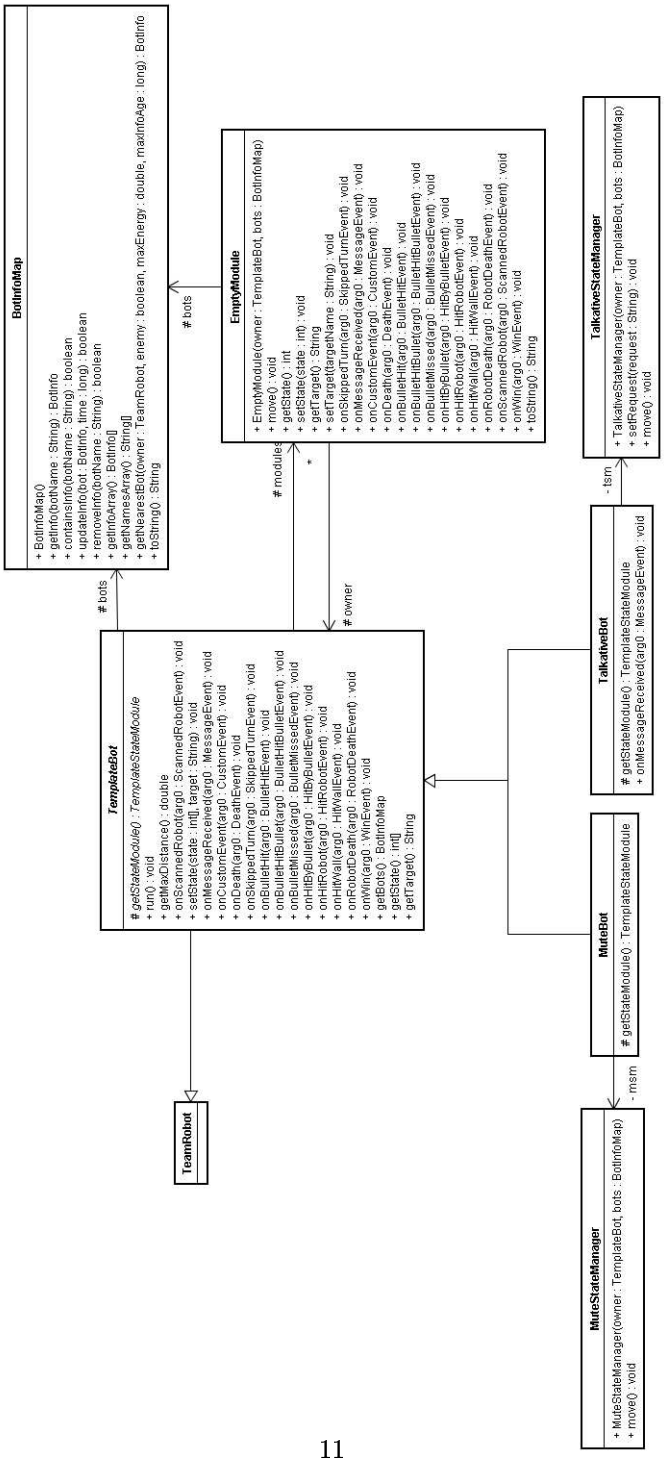


Figure 6: Modules package

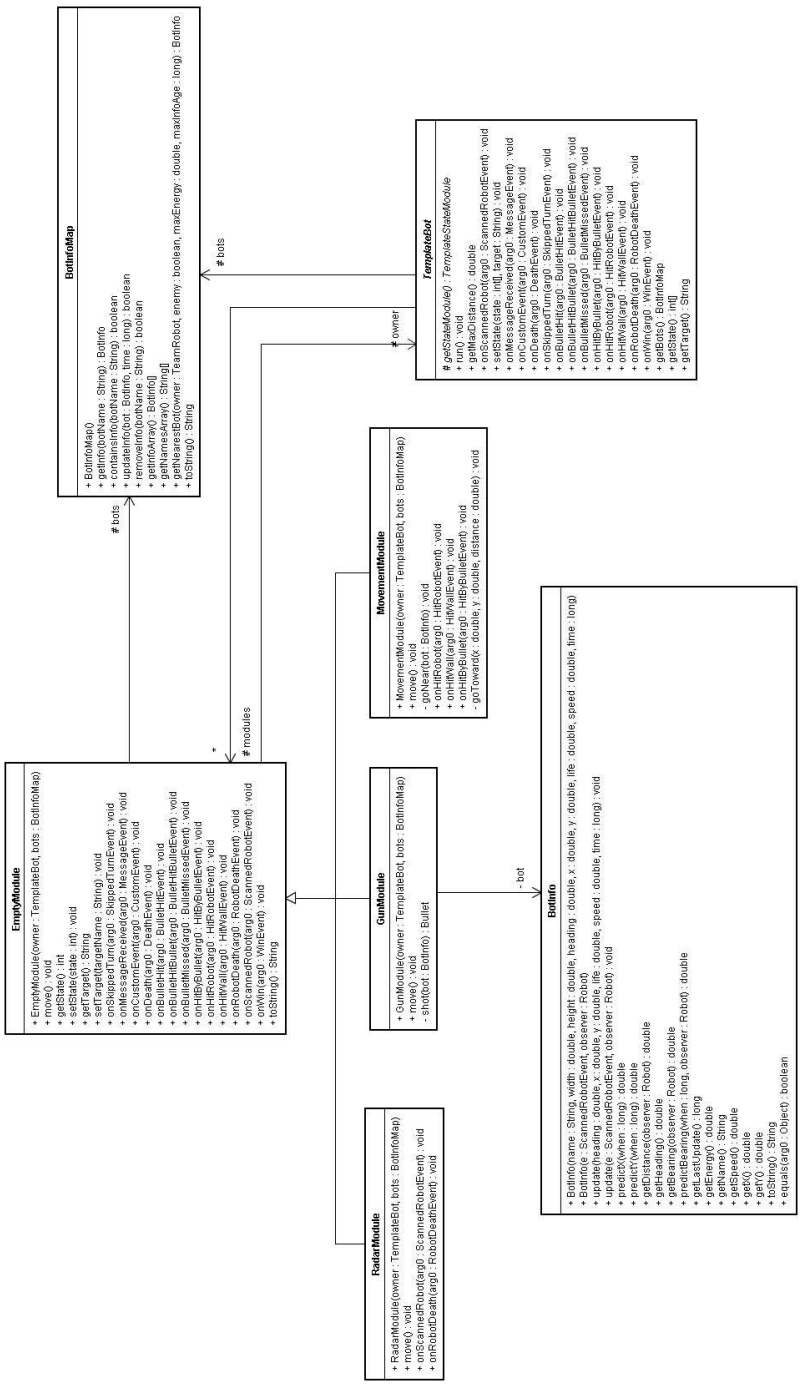
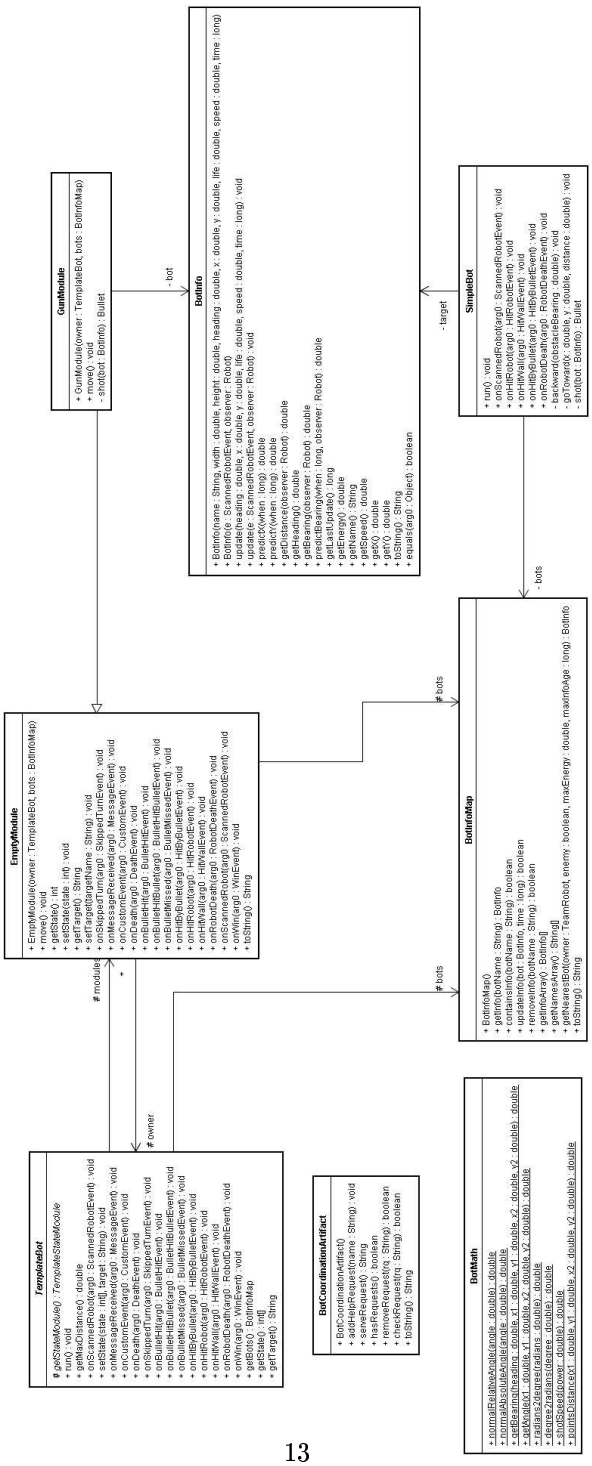


Figure 7: Utils package



entità ha senso quando guidata da una strategia chiara e di provata utilità. In questo particolare caso si può ipotizzare che sarebbe stato meglio partire con una squadra completamente non coordinata, constatarne eventuali problematiche e, a valle di queste osservazioni, integrare quanto già sviluppato con una strategia di coordinazione (magari basata su una divisione di ruoli). A conferma di quanto appena detto si è provato a realizzare una semplice squadra di bot dove non esiste nemmeno collaborazione implicita (in pratica è stato rimosso lo stato "DefendBot"), gli esperimenti hanno dimostrato che questa squadra compete assolutamente alla pari con le precedenti.

In conclusione si può affermare che, dove non esistono pattern strategici consolidati, è più utile avere un approccio costruttivo, iniziando da una struttura non coordinata e procedendo per piccoli miglioramenti (se esistono), abbandonando il preconcetto che la coordinazione sia, in ogni situazione, meglio del comportamento emergente.

References

[1] A. brooks. a robust layered control system for a mobile robot. ieee journal of robotics and automation, 2(1):14– 23, 1986.

[2] Robocode official web site, http://robocode.sourceforge.net.

[3] Michael Wooldridge and Nicholas R. Jennings. Intelligent agents: Theory and practice. HTTP://www.doc.mmu.ac.uk/STAFF/mike/ker95/ker95-html.h (Hypertext version of Knowledge Engineering Review paper), 1995.