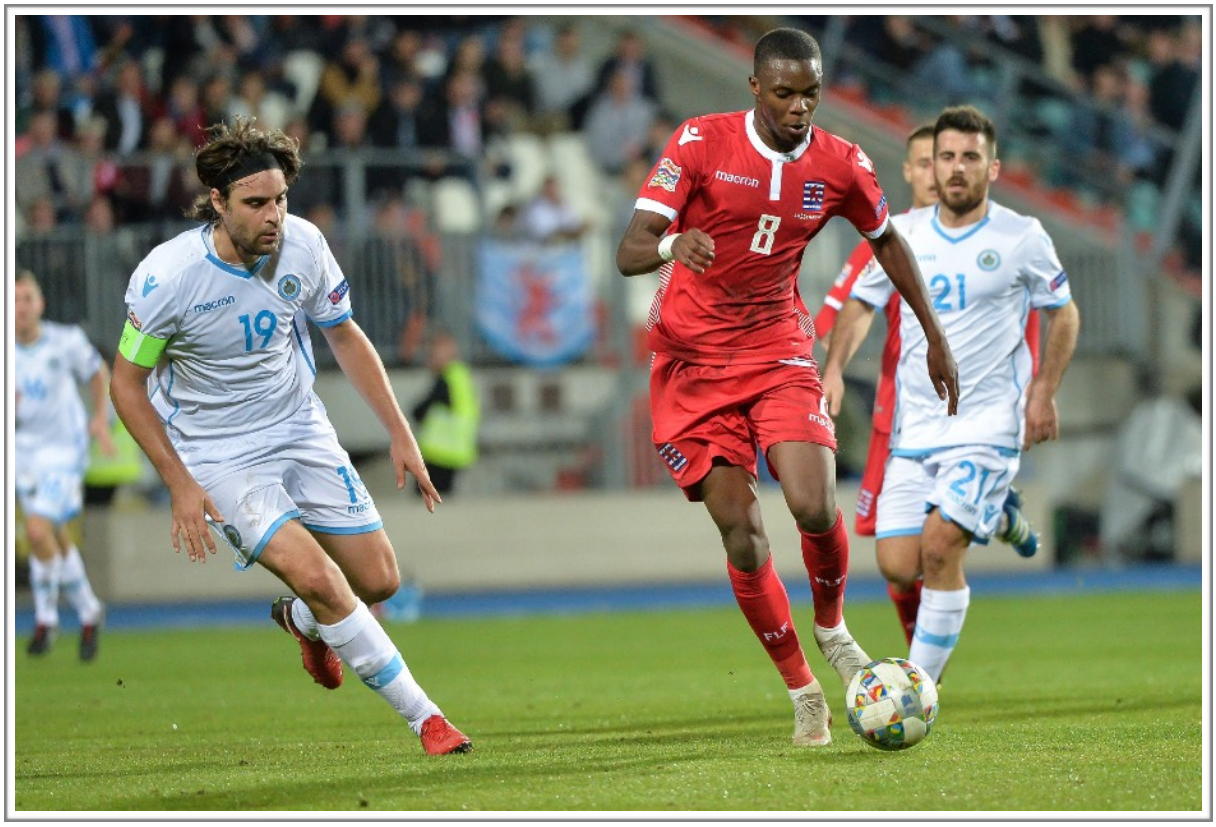
JAZN

Sistema multiagente per la simulazione di una partita di calcio



Nicola Giancecchi
m. 0900057545

Università di Bologna - Campus di Cesena
C.d.L. Magistrale in Ingegneria e Scienze Informatiche
Corso di Sistemi Autonomi

Rev. 1.0

Introduzione

L'intento di questo progetto per il corso di Sistemi Autonomi è quello di creare un semplice simulatore di una partita di calcio a 11 tramite l'implementazione di un sistema multiagente basato sul linguaggio Jason.
In questa relazione verrà fornita una breve introduzione sulla programmazione ad agenti, dopodichè si passerà all'analisi del caso di studio in oggetto e l'utilizzo degli agenti in tale implementazione.

Agenti

Il concetto di agente nasce dalla necessità di un paradigma apposito per la programmazione di entità autonome. Un **agente** è un sistema reattivo che possiede un certo livello di autonomia, ovvero può decidere autonomamente quale sia il modo migliore per compiere un determinato task in base alle proprie conoscenze e a quelle che riesce a ricavare dall'ambiente, col fine di raggiungere i propri *goals*.

La programmazione ad agenti, e più in generale la programmazione reattiva, si differenzia sostanzialmente dalla programmazione funzionale poiché il sistema viene eseguito a lungo termine, deve percepire l'ambiente continuamente ed agire di conseguenza in base a ciò che impara: non deve semplicemente prendere in input un dato, manipolarlo, fornirlo in output e terminare la propria esecuzione.

Gli agenti, infine, sono proattivi: sono stimolati dall'esterno, ma possono anche compiere azioni in base ai propri goal da raggiungere.

Sistemi multiagente

Abbiamo detto che gli agenti comunicano tra loro al fine di soddisfare i propri goals e che sono immersi all'interno di un ambiente. Questi due aspetti vanno a definire il concetto di **sistema multiagente** *(multi-agent system)*. In un sistema multiagente, ogni agente può soddisfare i propri goal tramite la conoscenza acquisita dagli altri agenti e dall'ambiente. L'agente ha una certa influenza sull'ambiente, la quale potrebbe sovrapporsi con quella di altri agenti.

Approccio BDI

Uno dei modelli più utilizzati nella programmazione ad agenti è *Beliefs-Desires-Intentions* (BDI), un modello che si avvicina al ragionamento umano:

- ***Beliefs*** rappresentano la conoscenza attuale dell'agente: messaggi dagli altri agenti, informazioni interne e percezioni dall'esterno. L'insieme dei belief compone la *belief base*.
- ***Desires*** rappresentano gli obiettivi che l'agente vuole perseguire
- ***Intentions*** rappresentano lo stato deliberativo dell'agente, ovvero ciò che l'agente ha scelto di fare.

Oltre ai componenti base, il modello BDI comprende anche:

- ***Plan library***, un insieme di sequenze di azioni (*recipes*) che possono essere eseguite dall'agente.
- ***Events***: trigger delle attività reattive degli agenti, es. aggiornamento dei beliefs, invocazione di plans o modifica degli obiettivi. Gli eventi vengono inseriti in una coda chiamata *event queue*.

Applicazione dei concetti

Giunti a questo punto ci si può chiedere: in che modo un sistema multiagente può rappresentare il caso di studio in oggetto?

Pensiamo al gioco del calcio, e in generale agli sport con la palla più diffusi (basket, volley, ecc.). Tali sport sono caratterizzati da un insieme di giocatori di squadre diverse, ognuno dei quali segue determinate regole e ha un proprio ruolo sul rettangolo verde, che devono raggiungere un obiettivo: segnare più punti possibile nel campo della squadra avversaria. Per raggiungere l'obiettivo, i giocatori si passano la palla tra di loro apprendendo ciò che sta accadendo attorno a loro e decidendo in poco tempo la soluzione migliore al fine di segnare un punto: tenere la palla, passarla o provare a fare punto. A volte tali scelte possono rivelarsi giuste e a volte sbagliate: facendosi segnare un punto in porta, buttando la palla fuori dal campo o facendosela intercettare dalla squadra avversaria. A completare il quadro ci sono l'arbitro e l'allenatore, i quali non sono in gioco ma possono prendere decisioni in base a ciò che sta succedendo sul campo, come annullare un gol o effettuare una sostituzione.

Si può notare a questo punto il rimando alle caratteristiche degli agenti. Entrambi, infatti:

- **decidono e deliberano autonomamente** in base alle proprie conoscenze;
- **sono sociali e comunicano** gli uni con gli altri;
- sono **situati** all'interno di un **ambiente** (il campo di gioco) dal quale possono captare informazioni che possono essere inserite tra le loro conoscenze.

Inoltre, tenendo a mente il paragone con la programmazione funzionale, i giocatori non sono semplici "macchine" che eseguono un compito. Durante i loro 90 minuti di gioco devono essere continuamente pronti ad agire nel caso ricevano stimoli dall'esterno o riescano a soddisfare i propri *beliefs*.

Dominio

Il progetto è volto a modellare una partita di calcio tra due ipotetiche squadre di calcio denominate *"Forlì City"* e *"Cesena United"*; in particolare:

- **i 22 giocatori in campo** (11 per squadra), ognuno con un ruolo e regole diverse da seguire (*portiere, difensori, centrocampisti, attaccanti*);
- **i 2 allenatori**, uno per squadra.
- **l'arbitro**.

Ogni componente è modellato tramite un agente del sistema che decide quali operazioni svolgere, in particolare:

- i **giocatori** hanno l'obiettivo di segnare nella porta avversaria, perciò cercano, per quanto possibile, di passare la palla in avanti. La palla, però, può essere intercettata da parte dell'altra squadra e portata verso la porta del loro avversario in qualsiasi momento;
- gli **allenatori** decidono se effettuare le 3 sostituzioni di giocatori a loro disposizione e spronano i giocatori;
- l'**arbitro** dà inizio alla partita, tiene traccia dei goal segnati e la conclude con un triplo fischio finale.

Implementazione

Il sistema è stato implementato utilizzando i linguaggi di programmazione *Java* e *Jason*, un'estensione del linguaggio ad agenti *AgentSpeak*.

Grazie a questo framework è possibile semplificare lo sviluppo di sistemi basati su agenti, favorendo la comunicazione tra agenti e la personalizzazione della loro architettura.

Inoltre è possibile interfacciare il sistema con Java, ad esempio per la definizione dell'ambiente o delle azioni interni. A tal proposito, l'ambiente di sviluppo che è stato utilizzato è Eclipse IDE 4.9 con plugin *Jason*.

Modellazione del sistema multiagente (MAS)

Il sistema multiagente è definito dal file `jazn.mas2j`, che specifica gli agenti iniziali del sistema, la classe relativa all'ambiente (*environment*) e la tipologia di infrastruttura (centralizzata, *Sadi* o *JADE*). Oltre a queste proprietà base, nel file possono essere inserite altre specifiche e parametri, richiamabili anche da Java.

```
MAS jazn {

      infrastructure: Centralised

      environment: env.JaznEnvironment

      agents:
            fcCoach coach;
            cuCoach coach;
            referee;
            forliCity player #11;
            cesenaUnited player #11;

      aslSourcePath:
            "src/asl";
}
```

Il sistema è così composto:

- infrastruttura **centralizzata**;
- classe environment `JaznEnvironment`, contenuta nel package `env`.
- agenti:
  - 2 agenti di tipo `coach` (`fcCoach` e `cuCoach`, rispettivamente per le squadre di *Forlì City* e *Cesena United*),
  - un agente `referee`,
  - 11 agenti `player` denominati `forliCity<1…11>`,
  - 11 agenti `player` denominati `cesenaUnited<1…11>`.

Modellazione degli agenti

Player

I giocatori sono i protagonisti della partita e ognuno possiede tre beliefs di base che li caratterizzano:

- `team({forliCity,cesenaUnited})` `// squadra di appartenenza`
- `jersey(<1…11>)` `// numero di maglia`

- `role({goalkeeper,defender,midfielder,forward})  // ruolo`

I giocatori agiscono ogni volta che uno di loro si impossessa della palla. La palla è in possesso del giocatore nel momento in cui gli viene aggiunto il belief `ball` da parte dell'ambiente. Dopodiché il giocatore può decidere se compiere un goal, nel caso il suo ruolo sia quello dell'attaccante (*forward*), oppure se passare la palla ad un suo compagno di ruolo diverso, ad esempio un difensore (*defender*) che passa la palla al centrocampista (*midfielder*).

Nel caso il giocatore voglia provare a segnare una rete, entra nell'achievement goal `tryGoal`. Il giocatore ha il 20% di probabilità di fare goal, calcolata casualmente tramite l'azione interna `.random`. In caso di goal, avvisa l'arbitro e dà la palla al centrocampista avversario, altrimenti avviene la rimessa dal fondo da parte del portiere avversario.

Nel caso il giocatore voglia passare la palla ad un giocatore di un altro ruolo, entra all'interno dell'achievement goal `passBall` relativo al proprio ruolo, il quale chiama l'azione interna `passTo` dell'ambiente stabilendo nell'unico parametro il ruolo di giocatore a cui passare la palla. La palla potrebbe essere anche intercettata da parte di un giocatore dell'altra squadra: in tal caso il possesso del belief `ball` passa all'altro giocatore.

La classe Java corrispondente a supporto dell'agente è `Player`, che implementa l'interfaccia `IPlayer`. Al suo interno sono memorizzate le caratteristiche del giocatore:

```
package model.impl;

import model.interfaces.*;

public class Player implements IPlayer {

      private Team team;
      private int jerseyNumber;
      private Role role;

}
```

Due enumeratori sono utilizzati in questa classe: `Team` e `Role`.

- `Team` rappresenta i due team, chiamati *`FORLI_CITY`* e *`CESENA_UNITED`* nell'enumeratore ma che contengono anche i rispettivi identificatori a livello di

agente (`forliCity` e `cesenaUnited`). È specificato anche il nome del coach ed è possibile ricavare l'opzione dell'enumeratore relativa all'altra squadra tramite la funzione `getOtherTeam()`.

- `Role` rappresenta i diversi ruoli disponibili - *`GOALKEEPER`*, *`DEFENDER`*, *`MIDFIELDER`* e *`FORWARD`*. Ogni ruolo viene inizializzato con la quantità di giocatori assegnati: in questo modo è possibile cambiare la formazione, che di default è impostata a **4-4-2**. È inoltre possibile sapere il ruolo "inverso" che servirà durante il passaggio della palla.

Coach

L'allenatore ha il compito di stimolare i giocatori e ha la possibilità di cambiare fino a 3 giocatori durante una partita. Tale possibilità è descritta dal belief `availableSwaps(3)`.

Il cambio di giocatore viene effettuato distruggendo e ricreando l'agente di un giocatore, nel caso ci siano ancora cambi disponibili e se la propria squadra abbia subito più di 2 goal.
Per effettuare uno scambio, l'allenatore viene invitato dopo ogni goal a controllare i cambi disponibili ed i goal ricevuti. Se le condizioni sono soddisfatte, viene decremento il valore del belief `availableSwaps`, dopodiché viene richiamata l'azione interna `coachAct.replaceAPlayer` che si occupa di trovare un giocatore da sostituire tra quelli della propria squadra. Il cambio avviene aggiungendo il belief `swap(X)` all'allenatore.

Il coach può intervenire ogni tanto per incitare i propri giocatori tramite il goal `ack`.

Referee

L'arbitro ha il compito di iniziare, terminare la partita e di tenere conto dei goal effettuati dalle squadre, informando di conseguenza gli altri agenti. Il punteggio viene tenuto tramite i due beliefs `fc_goals(0)` e `cu_goals(0)`, rispettivamente per le squadre di *Forlì City* e *Cesena United*, e viene chiamato tramite il goal `scored` relativo alla propria squadra. A questo punto, viene informato l'allenatore dell'altra squadra per effettuare eventuali cambi e viene stampato a video il risultato attuale.

La partita inizia al fischio dell'arbitro, tramite l'aggiunta del belief `whistled`, il quale si occupa di comunicare a tutti gli agenti che la partita è iniziata. La partita termina dopo 5 minuti tramite l'eliminazione, da parte dell'ambiente, di tale belief. Alla conclusione del match, vengono stampati a video i tre fischi finali assieme al

risultato. Il sistema multiagente viene in seguito chiuso dopo 10 secondi tramite l'azione interna `.stopMAS`.

Modellazione dell'ambiente

JaznEnvironment

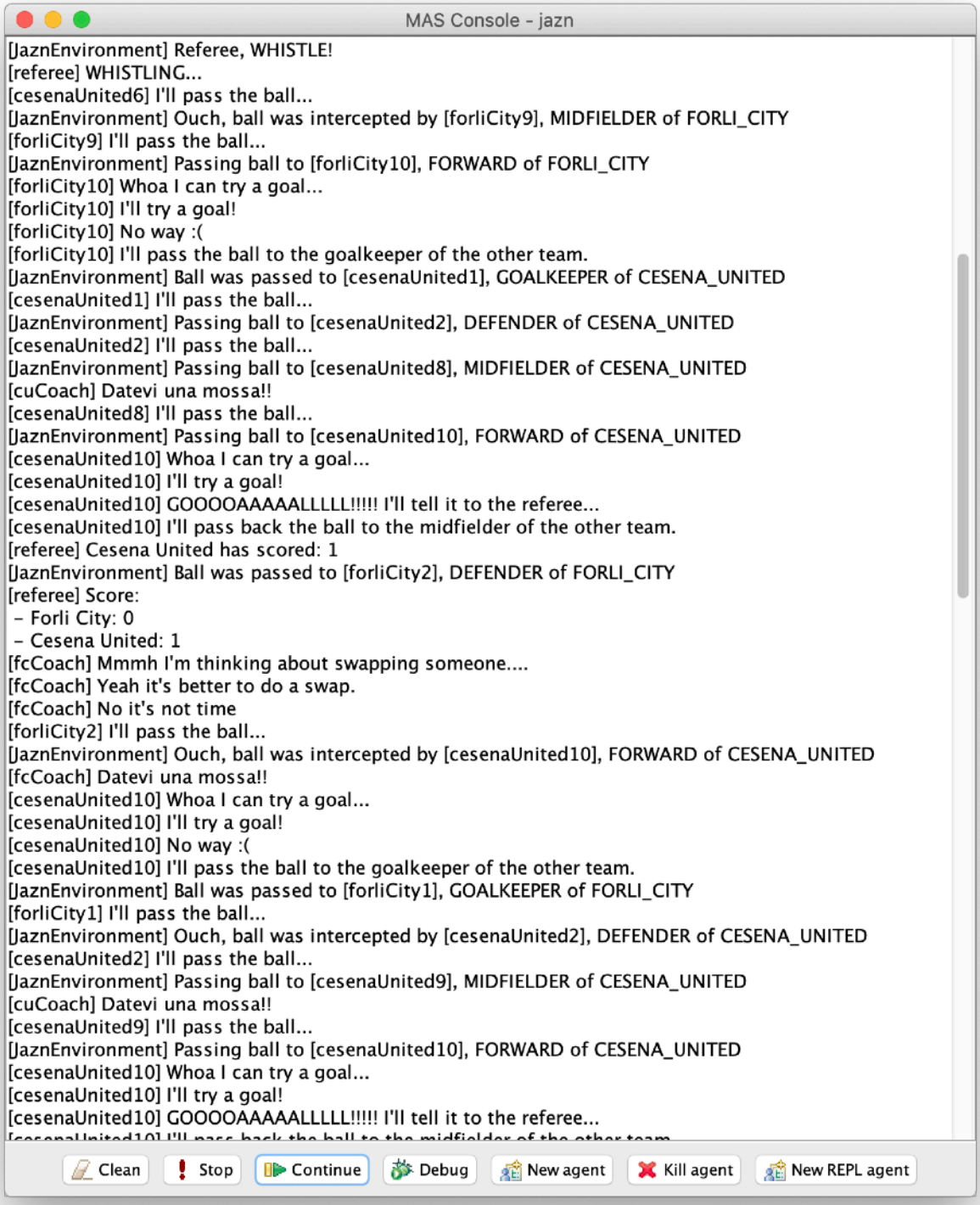
La classe `JaznEnvironment` viene utilizzata per le due operazioni che hanno bisogno di una conoscenza dell'ambiente circostante, ovvero la preparazione del campo da gioco e il passaggio della palla tra i giocatori. Possiamo pensare alla classe environment come al campo di gioco sul quale i vari agenti stanno giocando o partecipando.

Nella preparazione del campo (`prepareField`) vengono creati gli oggetti Java di tipo `Player` (interfaccia `IPlayer`) corrispondenti ai 22 giocatori in campo. Ogni oggetto, oltre ad essere creato associandogli la squadra, il ruolo e il numero di maglia, viene aggiunto alla struttura dati statica `players` composta da dizionari con chiavi di tipo `Team` e `Role`. In questa fase vengono anche aggiunti i beliefs `team(X)`, `jersey(X)` e `role(X)` di cui si è discusso in precedenza. Al termine della preparazione, viene scelto un attaccante di una delle due squadre al quale viene data la palla. A questo punto, l'arbitro è pronto per fischiare. Al fischio, viene salvato il timestamp attuale, così da poter terminare la partita dopo i 5 minuti previsti.

L'azione `passBall`, invece, permette di far passare la palla ad un compagno di squadra oppure di farsela intercettare da parte dell'avversario. L'intercettazione ha una probabilità del 40%, anch'essa generata casualmente, e in caso favorevole prevede che la palla passi ad un giocatore di ruolo inverso rispetto a quello personale: se il tiro è effettuato da un difensore, la palla passerà all'attaccante avversario; se invece è effettuato da un attaccante, verrà passata al difensore avversario. Come già accennato, il passaggio avviene tramite l'aggiunta del belief `ball`.

Esempio di partita

Vedi immagine nella pagina successiva.



Implementazioni future

Al fine di semplificare il dominio applicativo, non sono stati inclusi alcuni aspetti che potranno essere aggiunti in implementazioni future, come:

- il posizionamento dei giocatori nello spazio;
- rimesse laterali, calci d'angolo;
- rigori e punizioni;
- gestione di due tempi di gioco;
- ammonizioni ed espulsioni;
- implementazione dell'interfaccia grafica.

Riferimenti

- Codice sorgente del progetto: https://github.com/nicorsm/jazn
- Slides del corso: http://apice.unibo.it/xwiki/bin/view/Courses/Sa1718Slides
- R. H. Bordini, J. F. Hübner, M. Wooldridge, *Programming multi-agent systems in AgentSpeak using Jason*, 2007, John Wiley & Sons, Ltd.
- Documentazione delle API di Jason, http://jason.sourceforge.net/api/overview-summary.html
- Codice sorgente di Jason, https://github.com/jason-lang/jason

Foto di copertina © FLF / FSGC / UEFA Nations League