Giochi con apprendimento per sistemi ad agenti

Andrea Piran

Relatore: Prof. Raffaele Pesenti

Correlatore: Ing. Dario Bauso

Sommario

Il problema affrontato è l'apprendimento per sistemi ad agenti in ambienti di rete mediante un approccio basato sulla teoria dei giochi ed utilizzando come gioco di test l'*inventory problem* contro avversari che giocano in maniera aleatoria. In queste condizioni gli agenti hanno una conoscenza *a priori* limitata e, mediante ripetute partite di un gioco, ne ricostruiscono la struttura e scelgono la strategia da utilizzare. L'agente utilizza una stima dello stato medio degli agenti coi quali è connesso come dato di supporto alla decisione. La stima si basa sulla comunicazione della scelta fatta nella partita più recente dagli agenti. Viene mostrato come le prestazioni del sistema varino con la tipologia di connessione tra agenti, come la cooperazione tra agenti, mediante lo scambio di messaggi, consenta un comportamento migliore rispetto alla strategia mista e come il comportamento degli agenti si adatti a vari modelli aleatori della scelta strategica degli avversari.

Indice

Capitolo 1

Introduzione

In questo capitolo viene esposto in modo informale il problema affrontato ed introdotto il modello con le principali scelte di rappresentazione, infine vengono espressi i motivi d'interesse applicativo emersi dall'analisi della bibliografia.

1.1 Il problema di base

Uno dei problemi emersi di recente è il diffondersi di sistemi decentralizzati in molti contesti tecnici (e.g., cluster di calcolo, UAV, etc.) dato che un sistema centralizzato, a parità di prestazioni, generalmente può presentare costi più elevati e minore flessibilità. Un sistema decentralizzato, composto da unità indipendenti, presenta talvolta dei problemi dato che è necessario definire una coordinazione efficiente.

Una caratteristica desiderabile di queste unità è l'assenza di un eccessivo scambio di dati poiché se una grossa quantità di informazione facilita la coordinazione, una comunicazione che preveda un grosso scambio di messaggi ha un impatto sovente negativo sulle prestazioni del sistema.

Per studiare questo problema si è scelto di prendere come riferimento un problema noto in ambito logistico: l'*inventory problem*[1]. Dato che l'oggetto dell'analisi è un contesto distribuito che prevede le scelte di un certo numero di decisori si ritiene naturale studiarlo con un'approccio legato alla teoria dei giochi. Detta teoria ha come oggetto lo studio dei problemi legati alle decisioni contemporanee di n decisori (che vengono denominati *giocatori*) con l'obiettivo che ognuno di essi massimizzi la propria utilità.

1.1.1 Il problema delle scorte

Un azienda deve decidere se produrre un bene e metterlo in scorta, o non produrlo sapendo che in un caso o nell'altro deve sostenere dei costi. Nel caso in cui l'azienda decida di produrre il bene, si espone al costo di mantenimento in magazzino come scorte mentre, in caso decida di non produrre, si espone al rischio del pagamento dei costi relativi al trasporto del bene acquistato da terze parti ed alle penalità legate alla soddisfazione dell'ordine. Anche se il ricavo degli ordinativi è sufficiente a ricoprire i costi, l'andamento della domanda non è prevedibile comportando l'esposizione a queste spese. In altri termini, nel caso in cui l'azienda decida di produrre avrà costi nulli se arriva l'ordine poiché vengono coperte le spese di magazzino pari ad h, nel caso in cui non ci dovesse essere la commessa le suddette spese sono a carico dell'azienda. Se, invece, decide di non produrre non ci saranno costi di magazzino ma, in caso in cui arrivi un ordine, dovrà essere sostenuto un costo di trasporto $k/n(t)$, tanto più alto quanto minore è la possibilità di approvviggionarsi del bene da altre aziende.

I costi k ed h possono essere rappresentati come costanti positive ed $n(t)$ è il numero delle aziende che producono all'istante t, inoltre, si ha che se

[1]da qui in poi inventory viene tradotto come scorte come in [Hill94]

all'istante t arriva un ordine, al tempo $t+1$ l'azienda si espone al costo di trasporto ipotizzando che dopo un ordine le scorte siano nulle. Ogni azienda ha $n-1$ concorrenti di cui può conoscere lo stato all'istante $t-1$ con la condizione di avere dei rapporti commerciali.

Lo scopo del problema è ricavare qual'è la politica ottimale al fine di minimizzare i costi.

1.2 Aspetti del problema

Uno degli aspetti principali del problema è la presenza di un numero di decisori pari ad n, questo implica che l'utilità legata alla scelta di un decisore dipende *anche* dalle scelte di tutti gli altri decisori in un certo istante t. L'approccio d'analisi del problema sarà, quindi, basato sulla teoria dei giochi modellando l'azienda come un giocatore che persegue un proprio obiettivo, non necessariamente coincidente con quello degli altri.

L'utilizzo di una leva decisionale, che viene denominata *strategia*, è deciso da ogni azienda valutando, in maniera indipendente, i rischi. Questo porta a modellare un giocatore, dal punto di vista di una simulazione, come un *agente*. Questa scelta di modello è giustificata dalla considerazione che il paradigma della programmazione ad agenti prevede che ogni istanza dell'agente abbia dei comportamenti indipendenti.

Per modellare la comunicazione fra gli agenti sul loro stato si è scelto che gli n agenti connessi si scambino dei semplici *messaggi orali* [Lamp82], inoltre si assume che l'informazione su cui l'agente basa la scelta della sua strategia sia il proprio stato e lo stato medio degli agenti coi quali è connesso.

Un dato del problema è che la domanda non è prevedibile pertanto questa è modellata come un processo aleatorio che viene considerato in tre casi:

- La distribuzione di probabilità legata alla scelta di ordinare o meno è di tipo bernoulliano quindi *senza memoria*.
- La distribuzione di probabilità sia dipendente dalle scelte passate.
- La scelta sia costante a tratti in un intervallo diverso per ogni domanda.

Questa scelta è motivata dall'interesse di valutare come vari la dinamica della strategia dei giocatori per vari andamenti della domanda.

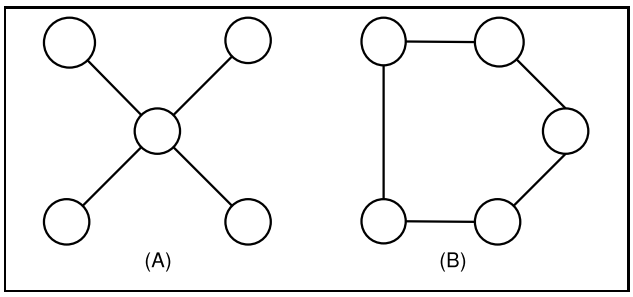

Figura 1.1: Esempi di connessione degli Agenti.

Rappresentando le relazioni tra i produttori per mezzo di un grafo, ci si pone il problema di voler valutare se esiste una dipendenza dalle proprietà di connessione del grafo ossia se il comportamento degli agenti è diverso se sono connessi come in fig1.1(A) rispetto alla connessione come in fig1.1(B) e come l'uscita del gioco ne dipenda posto che è preferibile che l'uscita sia di tipo non oscillante (a tratto continuo in fig.1.2) che tipo oscillatorio (tratteggiata in fig.1.2).

1.3 Agenti ed apprendimento

Una delle tecniche più interessanti per la programmazione di un agente è quella di associare il comportamento non a una codifica esplicita delle azioni e

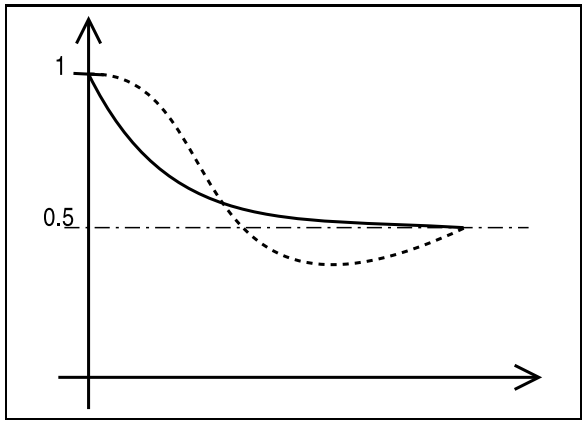

Figura 1.2: Uscita.

dell'ambiente in cui opera ma a quello che viene chiamato *apprendimento*. Ad ogni azione possibile per l'agente viene associato un peso che viene aggiornato dopo un certo intervallo di tempo in modo tale che venga aumentato di valore se il comportamento ha dei risultati positivi e diminuito in caso contrario. Dopo un certo numero di aggiornamenti, i pesi convergono ad un valore tale da ottenere il comportamento desiderato[2]. Inoltre viene anche usato come base per la ricostruzione di un ambiente non conosciuto. In termini di teoria dei giochi il peso può essere rappresentato dalla probabilità associata ad una strategia mista e l'intervallo di tempo per l'aggiornamento posto pari ad una partita e l'ambiente da riconoscere la struttura delle utilità associate alle strategie.

Un problema che ci si pone è la valutazione del comportamento degli algoritmi di apprendimento nei casi di probabilità relative alla dinamica della domanda.

[2]questa tecnica viene usata, ad esempio, per l'addestramento delle reti neurali [Nilss01].

1.4 Interesse applicativo

Di recente è stato studiato il problema del comportamento di un agente che opera in un ambiente che varia nel tempo e con poca conoscenza *a priori* dato che in queste condizioni l'apprendimento risulta una tecnica che risolve alcune difficoltà di una programmazione che tenga conto di tutti i possibili mutamenti dell'ambiente.

Un altro motivo d'interesse teorico ed applicativo consiste nel fatto che è stato poco trattato il caso in cui i giocatori devono scegliere le loro strategie non conoscendo, o conoscendo solo parzialmente (perché varia col tempo), la matrice delle utilità che è una modellazione dell'ambiente.

Inoltre si suppone che un approccio legato alla teoria dei giochi possa produrre dei benefici in applicazioni (come la gestione della congestione nel protocollo TCP [Fried98]) in cui gli algoritmi di soluzione legati a un puro approccio procedurale manifestano, in alcune condizioni, un comportamento non molto efficiente.

1.5 Struttura del documento

La tesi è organizzata nel modo seguente: nel secondo capitolo viene analizzata la bibliografia ed esposti i principali risultati teorici che servono come supporto al modello; nel terzo capitolo viene esposto il modello matematico del problema in esame; il quarto capitolo è dedicato alla trattazione della simulazione nei suoi aspetti implementativi; il quinto capitolo espone i risultati sperimentali; il sesto è dedicato alle conclusioni e all'analisi delle problematiche rimaste aperte. In appendice vengono riportate, per completezza, alcuni dettagli di progetto e la documentazione completa di classi e metodi della simulazione.

Capitolo 2

Bibliografia e risultati

In questo capitolo viene discussa la bibliografia utilizzata con lo scopo di esporre il contesto e i principali risultati utilizzati. Inoltre vengono esposti alcuni dei principali risultati della teoria dei giochi, introducendo la notazione che verrà utilizzata in fase di modello.

2.1 Analisi Bibliografica

L'analisi delle fonte bibliografiche è stata condotta con lo scopo di analizzare come in letteratura sono stati affrontati gli aspetti del problema esposti in sede d'introduzione.

2.1.1 Motivazioni

In [Fried98] viene considerato come caso di studio la rete Internet che per gli autori è un caso particolare di ambiente in cui devono essere distribuite delle risorse. Un problema di distribuzione di risorse, che include le scelte di un certo numero di decisori, è definito come la determinazione delle quantità di una risorsa in modo da soddisfare le necessità di coloro che la richiedono.

L'approccio al problema è basato sulla teoria dei giochi sottolineando come le soluzioni attuali al problema che hanno un approccio di tipo *best effort* (e.g., gestione della congestione nel protocollo TCP) portano, in alcuni casi, ad una distribuzione ineguale di dette risorse. La soluzione di questo problema comporta la trattazione di due problematiche: come definire l'ambiente in cui opera l'agente che ha delle risorse in comune e come l'agente può apprendere le caratteristiche di quest'ambiente.

2.1.2 Ambienti di rete

In [Grenn00] viene esposto quello che gli autori chiamano *network context*(i.e., ambiente di rete) dove gli agenti interagiscono attraverso l'uso di una risorsa come un canale di comunicazione e viene affrontato il seguente problema: ipotizzando che l'agente[1] abbia una conoscenza limitata dell'ambiente in cui opera (e.g., un utente internet non ha nessuna conoscenza della condizioni di carico della rete, l'unica cosa che vede è la velocità in cui scambia i dati.) ci si chiede attraverso quali meccanismi si possa arrivare ad un comportamento ottimale. Gli autori osservano come nella teoria dei giochi si assume generalmente che gli agenti siano a conoscenza della matrice dei payoff che implica una notevole conoscenza *a priori* non solo dell'ambiente ma anche di quanto fanno gli altri agenti, mentre in questo contesto l'agente conosce solo **il payoff relativo alla strategia scelta** e non è a conoscenza neanche di quanti siano gli agenti coinvolti nel gioco. In [Fried98] vengono presentate le condizioni in cui opera l'agente: ha informazioni limitate, non utilizza una procedura sofisticata di ottimizzazione, deve apprendere in che ambiente opera, generalmente non c'è sincronia di gioco e non è possibile

[1]Di qui in avanti si utilizza il termine agente per indicare sia il giocatore (in termini di teoria dei giochi) che l'agente inteso come implementazione.

individuare in modo semplice un'unità di tempo naturale, si osserva che non risultano rilevanti nè le azioni a breve termine nè quelle a lungo termine ma quelle a medio termine.

2.1.3 Apprendimento

Data la scarsa conoscenza *a priori* dell'ambiente l'agente deve apprendere la strategia ottimale da applicare attraverso un *algoritmo di apprendimento.*

Una caratteristica evidente di un'algoritmo d'apprendimento, prescindendo dal contesto, è che deve convergere alla strategia ottimale se l'ambiente è stazionario. In un ambiente di rete l'apprendimento deve avere due proprietà di base:

- *convergere*: La probabilità che un'agente utilizzi delle strategie non ottimali converge a zero dopo un certo periodo di tempo.
- *seguire la legge dell'effetto (Thorndike)*: Le strategie che hanno ottenuto dei payoff soddisfacenti in passato, vengono giocate con maggiore frequenza in futuro.

La definizione di questa classe di algoritmi discende anche dalla considerazione che, in questo contesto, gli agenti possono cambiare la loro strategia in ogni momento e in maniera indipendente, di qui la necessità che l'algoritmo di apprendimento non possa scartare delle strategie in modo definitivo. Se la struttura del gioco cambia nel tempo, strategie non ottimali in un certo istante t possono diventare ottimali con un ambiente cambiato.

L'algoritmo deve essere *reattivo* cioé le probabilità associate alle strategie devono convergere a un valore ottimale in un periodo di tempo breve in caso di una modifica dell'ambiente, questo comporta l'impossibilità di usare

algoritmi precisi ma computazionalmente complessi (e.g., Bayes) in quanto insoddisfacenti in queste condizioni.

2.1.4 Protocollo di consenso

In [Guer03] viene descritta l'idea di base del protocollo di consenso: definire un meccanismo avente lo scopo di permettere che un numero n di agenti arrivi ad una decisione condivisa. Viene anche precisato che è un caso particolare del problema dei generali bizantini [Lamp82]: ipotizzando di avere n generali di cui k traditori, trovare le condizioni per cui si possa ricavare dall'analisi dei messaggi quali generali siano dei traditori.

In [Olfa03], oltre alla definizione di un protocollo di consenso lineare per una rete di agenti, vengono esposte alcune delle proprietà del protocollo nel caso in cui il grafo che descrive le relazioni tra agenti abbia delle proprietà dipendenti dai valori che assume il laplaciano.

2.1.5 Grafo di Laplace

In [Guat99] viene esposta la relazione che esiste tra il tipo di connessione che ha un grafo e gli autovalori del laplaciano del grafo. Partendo dagli studi, principalmente condotti su λ_2 che è il secondo più piccolo autovalore della matrice di Laplace, viene mostrato come oltre ad essere usato per gli studi di convergenza ci sia una relazione esatta tra questo valore e la connessione dei nodi in un grafo.

Inoltre le proprietà del grafo di Laplace, trattate in [Mohar91], sono alla base di alcune dimostrazioni sulle proprietà del protocollo di consenso trattato in [Olfa03].

2.2 Risultati teorici di interesse

Allo scopo di snellire la trattazione del modello vengono riportati i risultati teorici che verranno usati nel seguito.

2.2.1 Definizione di gioco

Per *gioco* si intende una situazione in cui il risultato dipende non solo dalla propria scelta, ma principalmente dalla combinazione delle strategie scelte dagli avversari.

Un gioco G è definito come una coppia $< U, S >$ composta da:

- Un insieme di strategie indicato con S
- I pagamenti associati a queste strategie (funzione di utilità) indicato con U dove $U : S \rightarrow \mathbf{R}$.

Un gioco viene usualmente rappresentato in due forme:

- La forma bimatriciale o *normale* in cui sulla riga vengono riportate le strategie di un giocatore (denominato *giocatore riga*) e sulla colonna quelle dell'avversario (denominato *giocatore colonna*).
- Sotto forma di grafo dove ogni nodo che rappresenta una strategia di un giocatore ha come foglie le strategie dell'avversario.

Nella forma bimatriciale i pagamenti vengono rappresentati come una coppia di valori: *payoff del giocatore riga*, *payoff del giocatore colonna*, un esempio è in tab.2.1. Nella forma a grafo i pagamenti vengono rappresentati come peso degli archi, questa forma è riportata in fig.2.1 che modella il *dilemma del prigioniero*.

	Confessare	**Non confessare**
Confessare	$10, 10$	$1, 25$
Non confessare	$25, 1$	$2, 2$

Tabella 2.1: Forma normale di un gioco.

Il gioco può essere rappresentato come un grafo che ha come nodi radice le strategie di un giocatore e come foglie le strategie dell'avversario, il peso degli archi è dato dal payoff legato alle coppie di strategie scelte, come mostrato in fig.2.1.

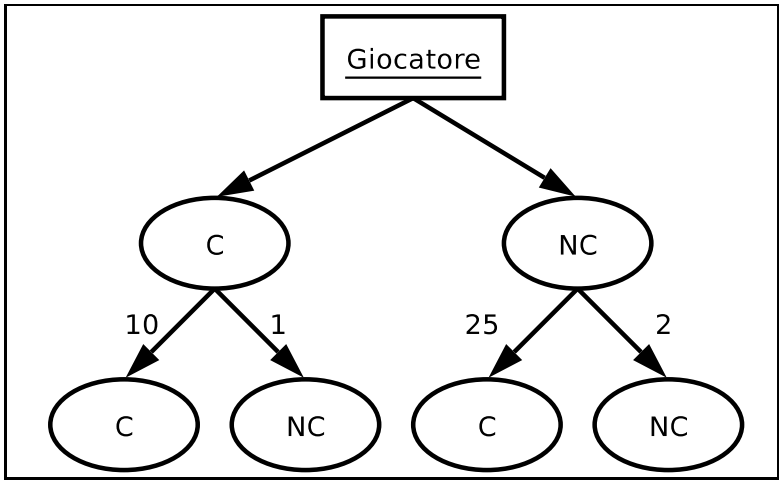

Figura 2.1: Forma a grafo di un gioco.

2.2.2 Strategie del gioco

Una *strategia* è una sequenza di azioni che viene effettuata al fine di massimizzare i propri benefici e/o minimizzare i propri costi e vengono definiti

due tipi di strategie:

Strategia pura Si ha una strategia pura se un giocatore sceglie come strategia di gioco una sola delle strategie possibili.

Strategia mista Si ha una strategia mista se il giocatore sceglie la strategia di gioco con un metodo probabilistico associando ad ogni strategia una probabilità.

2.2.3 Tipologie di gioco

Dato che il risultato di un gioco dipende dalle strategie scelte da tutti i giocatori si differenzia il tipo di gioco sulla base di come si modella l'atteggiamento di un giocatore rispetto ai suoi avversari. Si parla di giochi cooperativi o di giochi non cooperativi quando è presente una forma di comunicazione che permette ai giocatori di conoscere le azioni che compie il suo avversario consentendo una coordinazione.

Giochi cooperativi Un gioco si definisce cooperativo se tutti i giocatori hanno le stesse preferenze sull'insieme dei risultati.

Giochi non cooperativi Un gioco si definisce non cooperativo se i giocatori hanno preferenze diverse; nel caso in cui le preferenze siano opposte si parla di *giochi rigorosamente competitivi.*

2.2.4 Ottimalità di un gioco

Una sequenza di strategie $S_i = s_1, s_2, ..., s_n$ dove si denota con s_i la scelta del giocatore i è ottimale se massimizza un indice che rende conto dell'u-

tilità complessiva della società dei giocatori denominata Funzione di Scelta Sociale(SCF).

2.2.5 Soluzione del gioco

La soluzione di un gioco è l'insieme delle strategie scelte dai giocatori che sono ottimali. Vengono definiti dei tipi di soluzione in dipendenza dalla struttura del gioco e, di conseguenza, al metodo con cui viene valutata la soluzione che viene chiamata anche *equilibrio* dato che rappresenta quell'insieme di strategie che non muta nel corso delle partite.

Equilibrio di Nash

Un n-pla $(s_1, s_2, \ldots, s_n)^*$ è un equilibrio di Nash se nessuno dei giocatori ha un incentivo a modificare unilateralmente la propria strategia ovvero:

$$J_i(s_i, s^*_{-i}) \leq J_i(s^*_i, s^*_{-i}) \; \forall s_{-i} \in S_{-i}$$

si suppone che i giocatori scelgano le loro strategie contemporaneamente.

Soluzione per dominanza

Un n-pla $(s_1, s_2, \ldots, s_n)^*$ è una strategia che si definisce dominante se ha un'utilità migliore rispetto a tutte le altre strategie, qualunque sia la scelta dall'avversario ossia:

$$J_i(s_i, s^*_{-i}) \leq J_i(s^*_i, s_{-i}) \; \forall s_{-i} \in S_{-i}$$

L'insieme delle strategie che sopravvivono all'applicazione reiterata della soluzione per dominanza è denotato con D^∞ ed è l'insieme cui converge l'algoritmo d'apprendimento.

2.2.6 Tipi di aggiornamento

Nell'ambito della teoria dei giochi vengono definiti due tipologie di aggiornamento date le loro proprietà (in [Grenn00] è mostrato come da questo possa dipendere che l'equilibrio si sposti da uno di Nash ad uno di Stackelberg). È opportuno notare che non è necessario che in una simulazione tutti gli agenti debbano avere lo stesso tipo di aggiornamento.

Aggiornamento sincrono Si definisce *aggiornamento sincrono* la modalità d'aggiornamento per cui il giocatore aggiorna la sua strategia con frequenza pari a quella con cui gioca.

Aggiornamento asincrono Si definisce *aggiornamento asincrono* la modalità d'aggiornamento per cui il giocatore aggiorna la sua strategia con frequenza minore a quella con cui gioca[2].

2.2.7 Grafi

La modellazione della relazione tra i giocatori verrà fatta tramite un grafo pertanto si danno le principali definizioni.

Un modo per rappresentare le relazioni è l'utilizzo di un grafo; Un grafo è definito come $\mathcal{G} =< A, R >$ dove A è un insieme finito e non vuoto chiamato *insieme dei nodi di* $\mathcal{G}$ ed $R \subseteq RxR$ è l'*insieme degli archi orientati.*

$\mathcal{G}$ è *completo* se per ogni a_i e a_j esiste un arco da a_i a a_j.

Si definisce *cammino* una sequenza di archi che collega due nodi.

$\mathcal{G}$ è *a componenti connesse* se per ogni a_i e a_j esiste un cammino da a_i a a_j.

[2] È evidente che le probabilità possono essere aggiornate solo *dopo* aver giocato almeno una partita.

Grafo di Laplace Si denota con v un nodo del grafo, $V(\mathcal{G})$ l'insieme dei nodi del grafo e con $A(\mathcal{G})$ la matrice di adiacenza. Sia $d(v)$ il grado di $v \in V(\mathcal{G})$ e $D = D(\mathcal{G})$ la matrice diagonale con elementi $d_{vv} = d(v)$. La matrice $L = L(\mathcal{G}) = D(\mathcal{G}) - A(\mathcal{G})$ è chiamata *Matrice di Laplace del grafo* $\mathcal{G}$.

Capitolo 3

Modello del problema

In questo capitolo è sviluppato in modo formale il problema in esame e vengono messe in evidenza le problematiche che comporta. Il problema delle scorte è preso in considerazione in forma semplificata perché l'interesse piú che per il problema in sè[1] è per le problematiche strategiche degli agenti.

3.1 Definizione

Si ipotizza che n aziende producano un bene e debbano decidere se produrlo o meno in un certo momento t. Non potendo prevedere quando verrà fatto un ordinativo devono scegliere sapendo che:

- Se il bene verrà prodotto, in caso di mancato ordine, l'azienda dovrà sostenere un costo h per mettere in scorta il bene ed è un costo fisso.
- Se il bene non verrà prodotto l'azienda dovrà sostenere un costo di trasporto pari ad $k/n(t)$ approvvigionadosi da un'altra azienda.

[1] molto noto in letteratura tanto che il cap 12 di [Hill94] è interamente dedicato alla teoria delle scorte

- Negli altri casi il costo è nullo.

Ipotizzando che $k > h$ e dato che $n(t)$ è il numero di concorrenti che all'istante t hanno in giacenza il bene. L'azienda può conoscere le scelte dei concorrenti al tempo $t-1$ a condizione che sia in relazioni commerciali con loro.

Inoltre, se al tempo t c'è un ordine, al tempo $t+1$ l'azienda è nello stato di non produrre poiché si assume che, dopo una commessa, le scorte siano nulle comportando la necessità di approvvigionarsi all'esterno.

Interessa sapere qual'è la politica ottimale che minimizza i costi.

3.2 Modello del gioco

Si denota con $s_i(t)$ la scelta da parte dell'azienda della strategia i appartenente all'insieme S_i delle strategie, con $c_i(t)$ il costo associato e con c_{-j} il fattore che tiene conto dell'influenza della strategia $d_j(t)$ scelta dalla domanda. Col pedice negativo si denotano scelte e costi della domanda.

Il payoff J_i per l'azienda risulta:

$$J_i(s_i, s_{-j}) = c(s_i(t))c(d_j(t))$$

dove:

$$c(s_i(t)) = \begin{cases} \dfrac{k}{n(t)} & \text{se} \quad s_i(t) = \textbf{non produco} \\ h & \text{se} \quad s_i(t) = \textbf{produco} \end{cases}$$

$$c(s_{-j}(t)) = \begin{cases} 1 & \text{se} \quad s_i(t) = \textbf{non produco}, d_j(t) = \textbf{compro} \\ 1 & \text{se} \quad s_i(t) = \textbf{produco}, d_j(t) = \textbf{non compro} \\ 0 & \text{altrimenti} \end{cases}$$

Da queste equazioni si ricava la tabella dei payoff del gioco che è rappresentata in tab.1 .

Natura/Agente	**non produco**	**produco**
non compro	0	h
compro	$k/n(t)$	0

Tabella 3.1: Matrice dei payoff del gioco.

Dalla tab.3.1 si evince che il gioco non ammette soluzione tramite dominanza e, poiché la domanda non è deterministica, non ammette equilibrio di Nash.

La soluzione tramite dominanza esiste in condizioni di domanda costante poiché la tabella si riduce ad una matrice 1×2 e la soluzione diventa la coppia di strategie a payoff nullo (i.e., la coppia {*non compro,non produco*} e la coppia {*compro,produco*} in dipendenza della strategia della domanda).

Tenendo conto del fatto che l'azienda deve minimizzare i costi si può supporre che al variare di h ed k risulterà più conveniente utilizzare una strategia oppure un altra.

3.3 Modello del giocatore

L'azienda produttrice viene modellata come un giocatore il cui obiettivo è quello di minimizzare i propri costi. Si assume che, all'istante $t = 0$, la struttura del gioco non è conosciuta e, conseguentemente, sia nota al giocatore solo la leva decisionale rappresentata dalla scelta fra produrre e non produrre.

Inizialmente la strategia adottata dal giocatore è l'utilizzo di una strategia mista che ha l'effetto di permettere, conoscendo la scelta della domanda, di ricostruire il gioco secondo lo schema ad albero di fig.3.1. dove col nodo A è indicata la strategia del giocatore e con N è indicata la strategia della domanda.

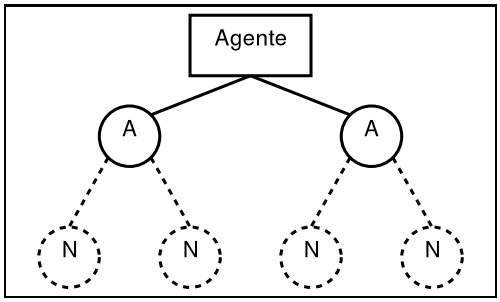

Figura 3.1: Gioco visto come albero.

I nodi vengono aggiunti quando viene giocata una coppia di strategie, quindi nelle prime partite il giocatore ha un albero di strategie incompleto come rappresentato in fig.3.2.

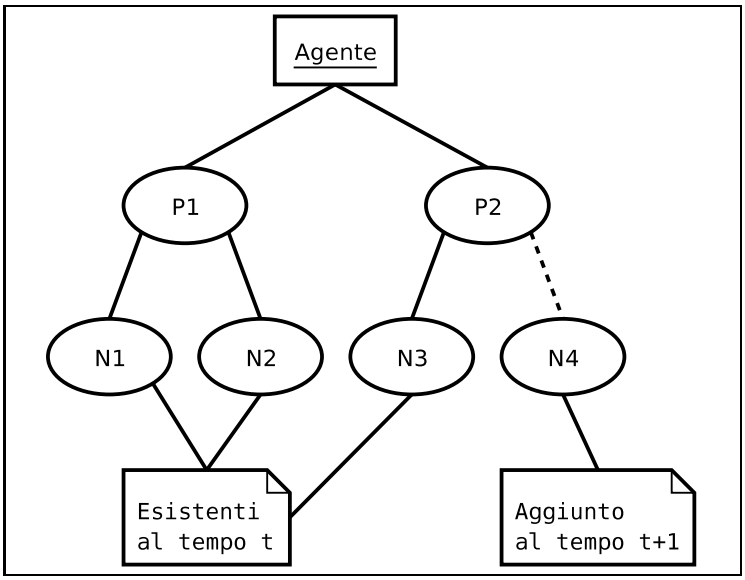

Figura 3.2: Ricostruzione del gioco.

In questo modo il giocatore, avendone le informazioni, ricostruisce il gioco allo scopo di determinare anche se esistono dei costi variabili e quali sono.

Si è scelto di applicare una strategia di tipo minimax che minimizza il

costo massimo relativo alle strategie i cui costi vengono pesati rispetto alla probabilità ottenuta tramite apprendimento (che verrà trattato più avanti) e rappresentata dall'equazione:

$$s_i = \min_i \left\{ \max_j \frac{c_{ij}}{p_i} \right\}$$

dove il pedice i è relativo alle strategie del giocatore e il pedice j a quello della domanda. Il peso relativo a p_i viene utilizzato in base a questa considerazione: ammesso che la strategia s_j abbia un costo massimo maggiore di s_i, il fatto che la probabilità s_i sia maggiore di s_j implica che *mediamente* il costo pagato è minore quindi ne risulta che la strategia s_j è una scelta migliore. Questa strategia è interpretabile come un criterio di minimo rischio dato che quella che viene scelta è la strategia col minimo costo massimo pesato rispetto alla probabilità.

Le probabilità associate alle strategie pure possono essere gestite con uno schema di questo tipo, indicando con p_i la strategia scelta:

$$p_i(t+1) = p_i(t) + f(J_i(s_i, s_{-i}))$$
$$p_j(t+1) = p_j(t) + f'(J_i(s_i, s_{-i}))$$

con la condizione che $\sum_{i \in I} p_i = 1$ e con $f()$ tale che aumentino le probabilità legate ai payoff favorevoli e diminuiscano le probabilità legate alle strategie non favorevoli ed $f'()$ tale da soddisfare la condizione che la somma delle probabilità dev'essere pari a uno.

Dal momento che un vincolo sul comportamento del giocatore è che se al tempo t c'è un ordine al tempo $t+1$ l'azienda non produce si ricava che il comportamento del giocatore è rappresentabile con un diagramma di sequenza che è mostrato in fig.3.3.

Dall'analisi dello schema si ricava che, nel caso in cui la domanda sia costante e richieda un prodotto, l'agente è forzato a rimanere nello stato di non

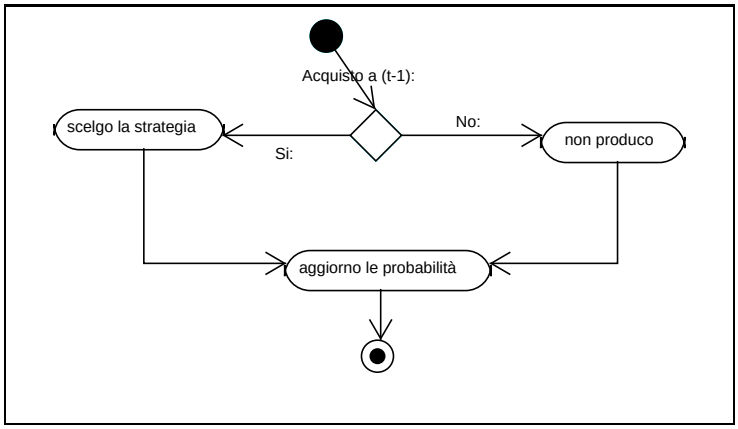

Figura 3.3: Schema del comportamento dell'Agente.

produrre, non valutando la strategia ma aggiornando soltanto le probabilità, e di sostenere un costo costante.

3.3.1 Apprendimento

L'apprendimento viene effettuato tramite l'algoritmo *RLA* (Responsive Learning Automaton) descritto in [Fried96] ed è un'estensione dell'SLA (Simple Learning Automaton) per ambienti di rete imponendo che ogni probabilità legata ad una strategia non sia minore di $\frac{\epsilon}{N-1}$ ed è costruito ponendo $p_i^0 = 1/N$ posto che N è il numero di strategie ed ϵ è una costante positiva che determina il grado di sperimentazione dell'agente ed aggiornando le probabilità secondo:

$$p_i^{t+1} = p_i^t + \gamma \pi_i^t \sum_{j \neq i} a_j^t p_i^t$$
$$p_j^{t+1} = p_j^t (1 - \gamma \pi_i^t a_j^t), \forall j \neq i \in N$$

con $a_j^t = \min\left[1, \frac{p_j^t - \epsilon/(N-1)}{\gamma \pi_i^t p_j^t}\right]$ e dove γ è un parametro da scegliere come compromesso tra velocità (per γ prossimo a 1) ed accuratezza (per γ

prossimo a 0) e π_i^t è la funzione che da il payoff al tempo t relativo alla strategia s_i.

Viene dimostrato che, in condizioni stazionarie, l'algoritmo converge all'insieme D^∞ (insieme delle strategie dominanti).

Poiché il costo $\frac{k}{n(t)}$ è variabile deve essere stimato sulla base delle scelte effettuate dagli altri giocatori che sono connessi.

In [Grenn00], dove vengono descritti vari algoritmi di apprendimento, viene puntualizzato come l'*RLA* non sia un algoritmo di tipo *a stadi* (un algoritmo a stadi divide un gioco in $\frac{1}{\gamma}$ partite e dopo ognuno di questi stadi aggiorna le probabilità legate alle strategie) ma sia un'*automa di apprendimento* ovvero aggiorna le probabilità alla fine di ogni gioco.

Si è scelto che viene comunicato lo stato in cui si trova l'agente e questo viene utilizzato come base per stimare i costi variabili facendo come assunzioni:

- Il giocatore sa che il costo variabile è inversamente proporzionale al numero di agenti che scelgono di produrre.
- La scelta di produrre viene associato ad un valore **1** che, oltre ad avere un significato di tipo logico ha anche un significato numerico.

Queste assunzioni portando ad ipotizzare che la precisione della stima del costo variabile dovrebbe dipendere dal grado di connessione dell'agente dato che all'aumentare di questo aumenta il numero di scelte che l'agente conosce.

3.3.2 Stima dei costi variabili

La stima viene effettuata tramite un *protocollo di consenso* lineare con ritardo, dato che viene trasmessa la strategia scelta al tempo $t-1$, definito dall'equazione:

$$u_i(t) = \sum_{j \in J_i} (x_j(t-\tau) - x_i(t-\tau))$$

In [Olfa03] viene dimostrato che, se ad ogni nodo di un grafo $\mathcal{G}$ *a componenti connesse* viene applicato questo protocollo distribuito, si ha che il vettore dei valori dei nodi x è la soluzione del sistema associato ad un sistema rappresentato da un'equazione del tipo

$$\dot{x} = -Lx$$

Dove L è il laplaciano del grafo $\mathcal{G}$. Inoltre, a regime, il sistema tende al valore medio. (i.e., Se $x^* = \lim_{t \to \infty} x(t)$, allora, $x_i^* = x_J^* = \text{Ave}(x(0)), \forall i, j, i \neq j$) se:

$$\tau \in (0, \tau^*) \ , \ \tau^* = \frac{\pi}{2\lambda_n} \ , \ \lambda_n = \lambda_{max}(L)$$

Viene inoltre dimostrato che questo sistema è un sistema stabile.

Dalla definizione di $u_i(t)$, denotato con $\tilde{n}(t)$ il valore stimato dei giocatori che producono e con N il numero dei giocatori connessi all'agente, si ricava che:

$$\tilde{n}(t) = u_i(t) - \frac{1}{N} x_i(t-\tau)$$

e si osserva che, in generale, $\tilde{n}(t) \neq n(t)$, dove $n(t)$ è il valore a denominatore del rapporto $\dfrac{k}{n(t)}$ che è il costo variabile del gioco, a meno che il grafo non sia completamente connesso.

Il payoff stimato $\tilde{\pi}_i$ può essere ricavato come

$$\tilde{\pi}_i = \frac{\pi_i^*(t)}{\tilde{n}(t)}$$

dove $\pi_i^*(t) = \dfrac{1}{M} \displaystyle\sum_{j=k}^{k+M} \pi_j^*, \ k \in [0, N_t]$, dove N_t è il numero di partite giocate al tempo t, ovvero il payoff osservato π_i^* associato alla strategia i è la media

delle ultime M osservazioni di questo. La media viene fatta per evitare due possibili problemi associati alla stima:

- Nel caso in cui il $\tilde{\pi}_i$ considerato fosse l'ultimo osservato potrebbe essere più basso di quello reale (i.e., se $n(t)$ è alto il payoff osservato è basso e viceversa).
- Nel caso in cui il $\tilde{\pi}_i$ considerato fosse il primo osservato potrebbe essere più alto di quello reale (e.g., potrebbe essere il valore di $\frac{k}{n(t)}$ per $n(t) = 0$).

3.3.3 Nota sull'implementazione dell'RLA

L'algoritmo *RLA* considera payoff positivi come guadagni da massimizzare, dato che il problema è la minimizzazione dei costi l'algoritmo viene applicato sul problema duale ottenuto scambiando le righe della matrice dei payoff quindi come problema di massimizzare i guadagni derivati dal risparmio sui costi.

La matrice dei payoff duale è riportata in tab.3.2.

Natura/Agente	**non produco**	**produco**
non compro	h	0
compro	0	$k/n(t)$

Tabella 3.2: Payoff duali del gioco.

Evidentemente, per la determinazione dei costi, deve essere riapplicata la stessa trasformazione per ricondursi al problema di partenza.

3.3.4 Differenza tra strategia mista e minimax

La strategia minimax fa si che il modello possa essere scritto in forma standard per un problema di ottimizzazione, però tenendo presente le motivazioni alla base dell'articolo da cui inizia l'analisi bibliografica [Grenn00] è opportuno precisare alcune cose.

L'approccio con cui gli autori partono per l'analisi dell'apprendimento in ambiente di rete è un di tipo non cooperativo dato che il tipo di conoscenza che raggiunge l'agente *non prevede la conoscenza della struttura del gioco* ma si limita alla conoscenza delle probabilità di successo legate alla strategia. Questo giustifica l'uso di una strategia mista in quanto questa prevede un'osservazione di tipo *trial and error* che fa a meno delle interazioni fra gli agenti ed anche con la domanda.

L'uso della strategia minimax prevede la conoscenza dei costi per ogni risultato del gioco (s_i, s_{-i}) quindi è necessaria la ricostruzione dell'albero delle strategie. Questa strategia è applicabile in contesti non cooperativi imponendo la conoscenza *a priori* dell'albero di gioco, modificando le ipotesi espresse in [Grenn00].

3.3.5 Complessità

Si denota con n il numero degli agenti, con m il numero di archi uscenti dall'agente e con p il numero di strategie dell'agente.

Il numero di messaggi scambiati nel sistema è $n \cdot m$, ipotizzando che m sia uguale per tutti gli agenti poiché ogni agente invia m messaggi per comunicare la strategia scelta all'istante $t-1$.

Il protocollo di consenso ha la stessa complessità poiché il numero di

messaggi ricevuti dall'agente coincide il numero di sommatorie da effettuare per il calcolo di u_i.

La strategia minimax ha complessità p^2 dato che richiede p accessi al nodo figlio per ogni nodo padre, assumendo che il numero di strategie dell'avversario sia pari a quella dei giocatori come nel gioco in esame.

Osservando che il numero di agenti è più alto del numero delle strategie, la complessità della strategia minimax è approssimabile con la complessità legata allo scambio di messaggi ed al calcolo del protocollo di consenso.

Da quanto osservato si ricava che, se la complessità della strategia mista non varia con la connessione (dipende dal numero p delle strategie del giocatore), la complessità dello scambio di messaggi e del protocollo di consenso varia in quanto m non è costante per tutte le connessioni: per la connessione a stella $m = n - 1$ per il centro stella e $m = cost. = 1$ per gli altri agenti, nella connessione a catena $m = 2$ per tutti gli agenti tranne per gli agenti che si trovano all'estremità della catena per i quali $m = 1$. Per la connessione completa, che è il caso peggiore, $m = n - 1$ per tutti gli agenti per cui si ottiene una complessita nell'ordine di $\mathrm{O}((n-1) \cdot (n-1)) = \mathrm{O}(n^2)$.

La complessità polinomiale legata alle connessioni comporta che c'è un aumento lineare dei tempi di calcolo legato all'aumentare del numero di agenti e permette simulazione con un numero relativamente elevato di agenti.

3.4 Modello della domanda

La domanda, dato che non è prevedibile, può essere rappresentata come un giocatore la cui scelta strategica è di natura aleatoria e comunica all'azienda la sua decisione di comprare o meno il bene da produrre. Questo consente all'azienda di potere valutare l'ambiente in cui si ritrova ad operare.

Il processo stocastico che modella la domanda, viene studiato in due casi: in un caso la domanda ha una probabilità di tipo bernouillano descritto da:

$$P\{X = 0\} = P_X(0) = 1 - p$$
$$P\{X = 1\} = P_X(1) = p$$

Nell'altro la domanda ha una probabilità di tipo *con memoria*:

$$P\{X = 0\} = P_X(0) = 1 - e^{-g(t)}$$
$$P\{X = 1\} = P_X(1) = e^{-g(t)}$$
$$g(t) = (1 - \alpha)g(t - 1) + \alpha x(t - 1)$$

Questa scelta è motivata dalla volontà di studiare il comportamento degli agenti nel caso in cui si pensa che risulti ininfluente una strategia che dipenda da un'analisi storica della domanda e, nel caso contrario, per apprezzarne le differenze.

Inoltre verrà pure simulato il caso in cui la domanda abbia un andamento costante a tratti (i.e., tipo un onda quadra) rappresentato in fig.3.4

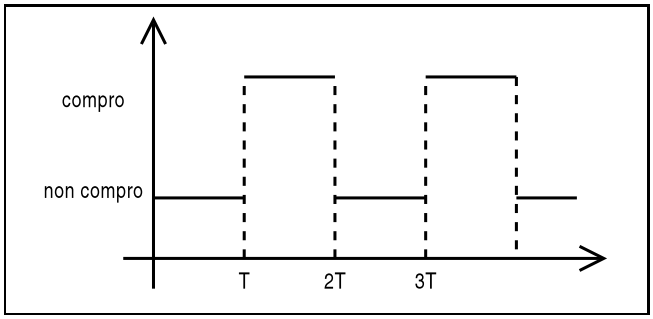

Figura 3.4: Andamento ad onda quadra della domanda.

3.5 Modello probabilistico dei giocatori

I modelli associati ai due giocatori seguono una distribuzione di tipo *binomiale*[2]. Per definizione, una distribuzione binomiale si ha quando il processo ha in uscita due eventi $\bar{A}$ ed A con probabilità rispettivamente $1-p$ e p. Se il processo è ripetuto n volte ed X è il numero di volte in cui si è ottenuto A allora la probabilità che X sia uguale ad A risulta:

$$P(X=k) = \binom{n}{k} p^k (1-p)^{n-k},\ k = 0, 1, \ldots, n,\ 0 < p < 1$$

con valore medio pari a $E(X) = np$ e varianza $\sigma^2 = npq$.

Il parametro p viene ricavato in modi diversi a seconda del tipo di giocatore.

Giocatore Nel caso del giocatore il parametro p è ricavato tramite l'algoritmo d'apprendimento *RLA* ed è in funzione del payoff ottenuto.

Domanda con memoria Nel caso della domanda modellata come un modello *senza memoria* il parametro p è costante e noto a priori per valutare, mediante simulazioni, il comportamento per vari valori e non dipende dalla storia delle scelte precedenti.

Domanda con memoria Nel caso della domanda modellata come un modello con memoria il parametro $p = e^{-g(t)}$ in modo tale da tenere conto della sequenza di scelte effettuate in passato ed evitare che si possano verificare lunghe sequenze di scelte costanti modellando una tipologia di domanda caratterizzata da una maggiore variabilità temporale.

[2]Scelta effettuata anche per ragioni implementative(cfr. 4.8).

Domanda ad onda quadra In questo caso non c'è un modello probabilistico ma un calcolo basato sull'ampiezza di ogni tratto a strategia costante.

3.6 Modello della comunicazione

Poichè un dato del problema è che i giocatori possono comunicare e scambiarsi le scelte all'istante $t-1$ si è scelto di rappresentarlo assumendo che sia una forma di cooperazione dato che tutti i giocatori hanno un interesse comune che è quello di minimizzare i costi, quindi per ogni giocatore si può supporre la preferenza per l'azione *produrre* rispetto a *non produrre*.

Inoltre vengono anche fatte le seguenti assunzioni:

- Ogni giocatore valuta individualmente le scelte dei giocatori adiacenti.
- Ogni giocatore ha un oppositore di tipo *Natura*[3].

La prima assunzione implica che all'aumentare dei giocatori connessi ogni giocatore ha una maggiore conoscenza media e viene fatta poichè non è detto che se il grafo che rappresenta i rapporti tra i giocatori non è completo tutti valutano la stessa media delle scelte.

La seconda assunzione viene fatta poichè semplifica parte del progetto dalla simulazione[4] poiché elimina la possibilità che la domanda, ipotizzando che il numero dei giocatori che modellano la domanda sia inferiore al numero dei produttori, debba muoversi da un produttore all'altro implicando la gestione di un caso di gestione dei payoff con domanda assente. Inoltre rende palese la tipologia di interazione tra *Agente*[5] e *Natura*.

[3]Da qui in avanti col termine *Natura* si intende il modello della domanda.

[4]Il modello, dal punto di vista implementativo, viene trattato nel capitolo seguente.

[5]Da qui in poi col termine *Agente* s'intende il modello dell'azienda produttrice.

3.6.1 Interazione Agente-Natura

Ogni agente di tipo giocatore gioca contro un agente di tipo Natura e la scelta di entrambi determina il payoff per il giocatore secondo quanto già esposto al par 3.2, l'informazione che ottiene l'Agente è la strategia utilizzata dall'agente di tipo Natura che viene utilizzata come foglia nell'albero che ricostruisce il gioco.

Per ragioni di semplicità implementativa lo scambio dei payoff, così come tutte le interazioni tra agenti, è mediato (l'implementazione è trattata nel capitolo 4.) tramite un agente che astrae gli aspetti legati alla comunicazione anche per garantire la separazione fra i due agente necessaria per le assunzioni sulla conoscenza che hanno i giocatori.

3.6.2 Interazione Agente-Agente

Gli agenti che sono connessi si scambiano informazioni sullo stato al tempo $t-1$ che può essere richiesto solo se esiste un arco che mette in relazione i due agenti. Questo comporta che non tutti gli agenti avranno le stesse informazioni e questo ne influenzerà le scelte strategiche.

Lo schema complessivo delle interazioni è mostrato in fig.3.5. dove con N è denotato l'agente di tipo natura e con P l'agente di tipo giocatore, con la linea continua è denotata la relazione fra agente ed agente e con quella tratteggiata quella fra agente e natura.

3.7 Funzione obiettivo

Ogni agente valuta il minimo del costo massimo relativo alle strategie (cfr. 3.3).

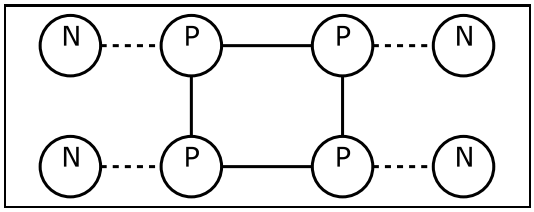

Figura 3.5: Interazione degli agenti.

Al fine di avere una rappresentazione intuitiva della bontà delle scelte strategiche è necessaria la valutazione di un indice che renda conto della dinamica dei costi. Si considera come indice il valore medio dei costi nelle partite dato che una semplice somma dei costi complessivi appare poco ragionevole in quanto aumenterebbe comunque visto che i pagamenti del gioco sono positivi. Indicato come $u(t)$ il costo associato alla scelta al tempo t e con N il numero di partite effettuate e con P il numero degli agenti, l'indice ha la forma:

$$F = \frac{1}{N} \sum_{i=1}^{P} \sum_{t=1}^{N} u_i(t)$$

Questo consente di valutare la strategia poiché una strategia ottimale farà si che il costo medio abbia un andamento che, ammesso che non tenda a zero ma ad un valore limite, sia decrescente.

Uno dei grafici in uscita della simulazione rappresenta l'andamento del costo medio.

Capitolo 4

Simulazione: architettura ed implementazione

In questo capitolo vengono descritti la piattaforma per la simulazione degli agenti e gli algoritmi di apprendimento utilizzati inoltre vengono discusse le principali scelte di progetto[1] del modello ad agenti.

4.1 Requisiti

In condizioni di scarsa conoscenza in [Grenn00] vengono esposte le caratteristiche degli agenti che sono:

- Gli agenti hanno una conoscenza *a priori* limitata del gioco.
- La funzione delle utilità cambia col tempo.
- L'apprendimento non è influenzato dall'utente.
- In giocatori possono aggiornare le caratteristiche in modo asincrono.

[1] In appendice viene riportato il progetto nei dettagli

mentre le caratteristiche di un algoritmo di apprendimento sono:

- In un ambiente stazionario l'agente gioca la strategia con l'utilità piú alta
- L'agente deve accorgersi di modifiche dell'ambiente
- La probabilità di qualunque strategia dev'essere sempre maggiore di una costante fissata.
- l'algoritmo deve reagire ai cambiamenti nell'ambiente in un tempo ragionevolmente breve [Fried96][2].

4.2 Piattaforma di simulazione

Come piattaforma è stata scelta RePast[3] (http://repast.sourceforge.net) che è una piattaforma per simulazioni ad agenti che usa il linguaggio Java. Supporta la modellazione di agenti con un modello sostanzialmente di tipo *BDI* (credenze, obiettivi, azioni) ed è stata scelta poiché, come evidenziato in [RP03], è una piattaforma pensata con riferimento a problematiche proprie della teoria dei giochi (i.e., l'agente è pensato come un attore sociale). La parte legata al modello probabilistico e agli aspetti legati all'algebra matriciale vengono gestiti utilizzando la libreria Colt (http://cern.ch/hoschek/colt) che è una libreria pensata per problematiche di calcolo numerico.

La classe Random (uchicago.src.sim.util.Random) della libreria RePast incapsula la generazione dei numeri casuali della libreria Colt

[2]Questo implica l'impossibilità di utilizzare algoritmi basati sul test di Bayes per il costo computazionale di questi.

[3]REcursive Porous Agent Simulation Toolkit.

(cern.jet.random.*) con una singola classe con funzioni di *wrapper* in modo da ottenere una gestione semplificata delle distribuzioni probabilistiche e di assicurare che tutte le distribuzioni usino la stessa successione di numeri casuali. Il dettaglio del funzionamento è al paragrafo 4.11.

4.3 Architettura delle classi

Dall'analisi dei requisiti si ricava il diagramma delle classi in fig.4.1 dove *GameSimModel* è la classe principale della simulazione che istanzia i due tipi di agenti *Agent* il cui ruolo è quello di giocatore e *Chance* che ha il ruolo della domanda.

La classe *Agent* è in relazione con la classe *PlStrategy* che incapsula le funzionalità legate alle scelte strategiche, per questo quest'ultima classe instanzia oggetti della classe *Consensus* che incapsula i metodi per le funzionalità legate alla coordinazione e della classe *Learning* per gli algoritmi di apprendimento.

La classe *Chance* implementa il ruolo della domanda ed è legata alla sola classe *NtStrategy* dato che il suo modello per la scelta delle strategie è aleatorio.

Le relazioni fra le classi sono del tipo 1-1 tranne la relazione tra *GameSimModel* e le classi *Agent* e *Chance*, la relazione tra *Agent* e *PlStrategy*, e la relazione tra *Chance* e *NtStrategy* che sono del tipo 1-n.

Le classi *PlStrategy* ed *NtStrategy* sono sottoclassi della classe base *GenericStrategy* che modella la strategia generica di un giocatore e non contiene l'attributo *payoff* che non è d'interesse per l'agente che modella la domanda e nemmeno l'attributo *memory* che in caso di valore *true* indica che il modello probabilistico è con memoria.

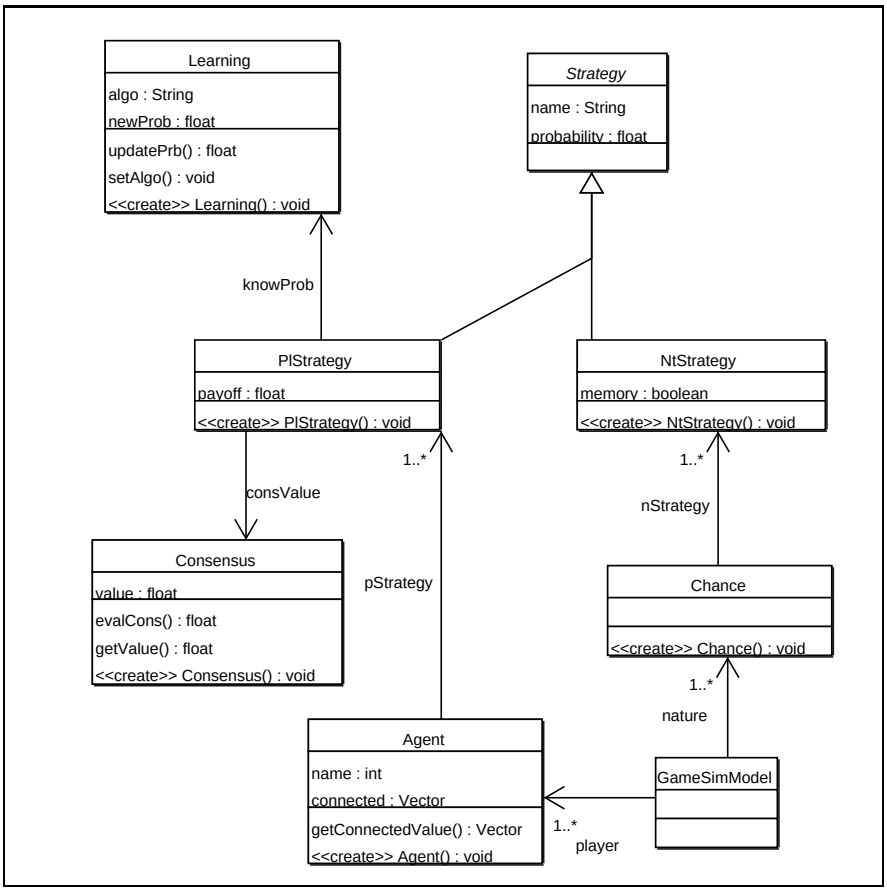

Figura 4.1: Diagramma delle classi di analisi.

4.4 Modello dell'Agente

L'agente, implementato nella classe **Agent**, modella dal punto di vista della simulazione un giocatore che sceglie l'azione da compiere secondo una strategia minimax sul costo legato alla strategia pesata rispetto alle probabilità (inizialmente uguali) che vengono modificate sulla base dei risultati ottenuti nei giochi precedenti.

In caso in cui la strategia minimax non può essere determinata (i.e., all'inizio della simulazione in cui i payoff non sono noti) viene usata una strategia mista.

Un caso particolare si ha quando in t lo stato natura è *compro* poiché l'agente in $t+1$ è forzato ad assumere lo stato *non produco*.

Il metodo **Agent.play()** è il metodo che viene chiamato affinché la sequenza di azioni del giocatore sia consistente con quanto espresso in fase di modello dei giocatori.

Le probabilità associate alla strategia vengono determinate tramite apprendimento.

4.4.1 Apprendimento: RLA

L'algoritmo **RLA** viene calcolato da ogni giocatore in maniera indipendente, **ag** è l'agente di tipo **player** di cui il cui metodo **getProbability(i)** restituisce la probabilità associata alla strategia **i**. Di seguito è riportato il codice Java che calcola la probabilità aggiornata.

```
for (int i = 0; i < N; i++)
    if (i != selected)
        {
            float tProb = ag.getProbability(i);
            if (tProb != (epsilon / (N - 1)))
```

```
                ai = Math.min(1, ((tProb - (epsilon / (N - 1)))
                                / (gamma * result * tProb)));
            else
                ai = 0;
            fixed = fixed + (ai * tProb);
        }

newProb = oldProb + gamma * result * fixed;
ag.setProbability(newProb, selected);

for (int i = 0; i < N; i++)
    if (i != selected)
        {
            oldProb = ag.getProbability(i);
            newProb = oldProb * (1 - gamma * result * ai);
            ag.setProbability(newProb, i);
        }
```

La variabile intera *selected* rappresenta la strategia scelta al tempo t dall'agente e viene usata per identificare il nodo da scegliere nella lista delle strategie. Viene calcolata prima la probabilità della strategia scelta, poi vengono calcolate le probabilità per le altre strategie.

4.4.2 Ricostruzione del gioco

L'agente, sulla base delle informazioni che riceve[4], ricostruisce il gioco rappresentandolo internamente come un albero il cui nodo sorgente è la strategia del giocatore ed i nodi figli sono le strategie dell'agente **Chance**. Dal punto di vista implementativo viene usata la classe standard Java *Vector* ed ad ogni

[4] Questo aspetto verrà trattato nella parte relativa alle interazioni al par 4.6.1

nodo viene associato il payoff corrente che è quello della coppia di strategie (s_i,d_j).

Ad ogni strategia dell'agente **Agent** viene aggiunto un nodo in questo modo:

```
if (!found)
   {
      PlStrategy current = (PlStrategy) lstStrategy.get(getStrategy());
      current.addOppStrategy(oppStrategy, strategy.getPayoff());
   }
```

dove *found* è una variabile booleana che ha valore *true* se la strategia è stata trovata nella ricerca nell'albero delle strategie.

4.4.3 Stima del costo variabile

Il calcolo del valore dato dal *protocollo di consenso* è incapsulato nel metodo **Agent.getCons** che viene usato come base per la stima del costo variabile secondo questa formula:

```
for (int i = 0; i < N; i++)
   {
      xj = (Integer) lstCons.get(i);
      value = value + xj.intValue() - xi;
   }
```

Il valore del protocollo è calcolato in maniera iterativa sommando i valori delle somme $x_j - x_i$ per tutti i valori x_j. Il numero N delle somme è pari al numero di giocatori connessi ed è coincidente con la dimensione della lista **Agent.lstConn**.

4.4.4 Gestione della connessione

La classe **GameSimModel** a partire dal valore ritornato dalla lista delle connessioni possibili ha la responsabilità di settare la connessione degli agenti che viene memorizzata nella lista **Agent.connected** secondo uno schema di questo tipo (l'esempio è per la connessione completa):

```
case 2 : // COMPLETE
for (int i = 0; i < P; i++)
     {
        ag = (Agent) player.get(i);
        j = 0;
        while (j < P)
            {
                if (j != ag.getName().intValue())
                    ag.setConnectedValue(new Integer(j));
                    j++;
            }
    }
break;
```

Ogni agente ha come attributo una lista **Agent.lstConn** i cui nodi sono i nomi degli agenti a cui è connesso (ogni agente ha come nome un numero progressivo). Per ogni agente, all'inizio della simulazione, vengono aggiunti i nodi alla lista in base al tipo di connessione da usare nella simulazione (e.g., se la connessione è a stella, il centro stella avrà come lista i nomi di tutti gli altri agenti, mentre quest'ultimi avranno come unico nodo della lista il centro stella.).

4.5 Modello della domanda

Un altro tipo di agente modella la domanda, la differenza del ruolo, che giustifica l'implementazione come un diverso tipo d'agente, consiste nella diversa strategia di gioco la cui probabilità legata alla scelta delle azioni è di un fenomeno aleatorio.

Siccome l'agente che ha il ruolo di avversario ha una strategia di tipo aleatoria la probabilità associata alle strategie viene aggiornata secondo quanto espresso in sede del modello del problema (par.3.4) non essendo necessaria una valutazione del payoff.

La strategia ad onda quadra ha quattro modalità:

- *Equal*: Ogni onda dell'agente **Chance** ha la stessa ampiezza.
- *SubMult*: Le onde sono sottomultiple dell'ampiezza di base.
- *Mult*: Le onde sono multiple dell'ampiezza di base.
- *Random*: Le ampiezze sono casuali.

4.6 Classi assenti nel modello d'analisi

Per ragioni di carattere implementativo sono state inserite delle classi assenti in fig.4.1 che vengono di seguito documentate.

4.6.1 Classe Mechanism

La classe *Mechanism*, assente nel diagramma delle classi d'analisi di fig.1 è implementato per gestire due aspetti della simulazione: la gestione della comunicazione dei payoff ai giocatori e la gestione delle interazioni tra gli agenti facendo da astrazione sul meccanismo di comunicazione.

La sequenza d'interazione fra agente di tipo **Agent** ed agente di tipo **Chance** è rappresentata in fig.4.2. Lo schema è lo stesso, con le opportune modifiche, anche per l'interazione fra gli agenti di tipo **Agent**.

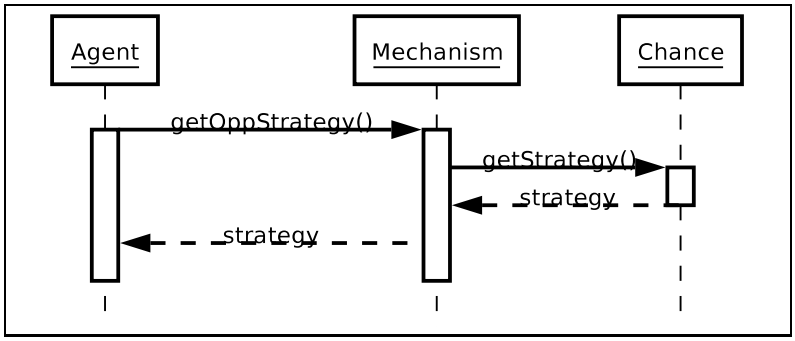

Figura 4.2: Interazione fra gli agenti.

4.6.2 Classi AGEdge e AGNode

Le classi *AgEdge* e *AgNode* sono realizzazioni dell'interfaccia *Edge* e *Node*. Per la realizzazione del display degli agenti, che è descritto nel seguito, la libreria utilizza un modello basato su un *social network* quindi ogni agente, per essere rappresentato sullo schermo, deve essere un nodo e deve poter possedere un arco che lo collega agli altri agenti.

4.6.3 Classe OppStrategy

La classe *OppStrategy*, che estende la classe astratta *GenericStrategy*, è un'astrazione sulla strategia dell'avversario (i.e., la domanda) che viene usata per la ricostruzione dell'albero di gioco. Ogni oggetto di questa classe viene inserito in una lista attributo dell'oggetto di classe *PlStrategy*.

4.7 Gestione degli agenti

Gli agenti vengono inseriti in una lista e gestiti nel metodo **GameSimModel.step** secondo questo schema:

```
for (int i = 0; i < numPlayers; i++)
{
    nat = (Chance)nature.get(i);
    nat.play();
    ag = (Agent)player.get(i);
    ag.play();
}

fictPlayer.play();

for (int i = 0; i < numPlayers; i++)
{
    ag = (Agent)player.get(i);
    ag.setGamePayoff();
}
```

La prima procedura iterativa serve per fare giocare l'agente di tipo **Chance** e quello di tipo **Agent**. L'agente di tipo **Mechanism** gioca dopo gli altri agenti per garantire che il calcolo del costo variabile $\frac{k}{n(t)}$ avvenga dopo che tutti gli agenti col ruolo di giocatore hanno fatto la loro scelta strategica ottenendo la correttezza del calcolo del parametro. Inoltre il metodo **Agent.setGamePayoff** è utilizzato per settare la strategia scelta al tempo $t-1$ ottenendo la certezza che la strategia comunicata durante la prima iterazione sia quella corretta, e non quella al tempo t come sarebbe possibile se l'aggiornamento dell'attributo **Agent.oldStrategy** venisse fatto nella prima iterazione.

4.8 Modello probabilistico

Il modello probabilistico viene implementato utilizzando i metodi forniti dalla classe *Random* che vengono utilizzati secondo questo schema:

```
Random.createBinomial(1, prob); // nel costruttore
...
Random.binomial.setNandP(1, prob); // nel metodo opportuno
st = Random.binomial.nextInt();
```

Il metodo **createBinomial**, che si trova nel costruttore della classe, genera la sequenza dei numeri di base. Nel metodo in cui dev'essere usata la scelta mediante il modello probabilistico, il metodo **setNandP** della classe *Random* aggiorna la probabilità p del modello binomiale e **nextInt** restituisce un intero con le probabilità settate precedentemente.

Un caso particolare si ha con la generazione della strategia ad onda quadra generata in base all'ampiezza. Nel caso random l'ampiezza delle strategie viene generata mediante un modello probabilistico uniforme con estremi da zero a un numero pari a N (numero dei giocatori) volte l'ampiezza di base.

```
/* nel metodo GameSimModel.manageNtStrategies */
Random.createUniform(0, baseAmp * numNature);
   for (int i = 0; i < numNature; i++)
     {
        nt = (Chance) nature.get(i);
        int k = Random.uniform.nextInt();
        nt.setAmplitude(k);
     }
```

Il numero k ottenuto viene quindi usato come ampiezza dell'onda usata dall'agente, e questo procedimento viene iterato per tutti gli agenti.

4.9 Tipi di connessione

Le connessioni gestite dalla simulazione sono state scelte tenendo conto di due fattori:

- Le connessioni possono essere rappresentate da un grafo *a componenti connesse* per l'applicabilità del protocollo di consenso
- I seguenti grafi risultanti dalle connessioni prese in considerazione rappresentano configurazioni comune di reti (i.e., si pensa sopratutto alle configurazioni delle reti di elaboratori)

Connessione a catena La connessione a catena, rappresentata in fig.4.3, è una delle connessioni più semplici ed è stata scelta poiché è la connessione col grado più basso.

Per via del basso grado di connessione (ogni agente ha al massimo due dati, e i nodi alle due estremità ne hanno uno solo) si ipotizza che sia la configurazione che darà le peggiori prestazioni.

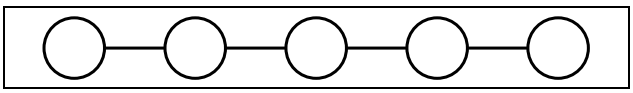

Figura 4.3: Connessione degli agenti in linea.

Connessione ad anello La connessione, rappresentata in fig.4.4, è una connessione di grado 2 uguale per tutti i nodi.

Si ipotizza che abbia prestazioni migliori della connessione a catena.

Connessione a stella La connessione, rappresentata in fig.4.5, presenta una particolarità: il nodo centrale (*centro stella* o *radice* a seconda dei conte-

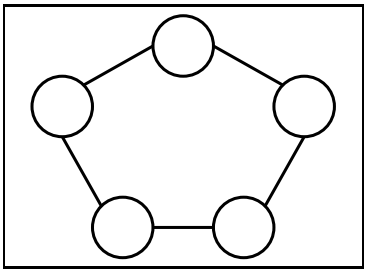

Figura 4.4: Connessione circolare degli agenti.

sti) ha l'informazione completa degli agenti a cui è connessa, mentre gli altri nodi hanno come unica informazione la strategia del nodo sorgente.

Si ipotizza che abbia prestazioni dipendenti dalla scelta strategica del nodo radice.

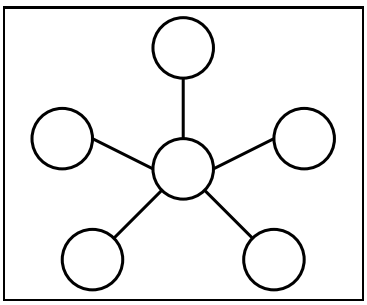

Figura 4.5: Connessione degli agenti a stella.

Connessione completa La connessione, rappresentata in fig.4.6, è la connessione di grado massimo e tutti gli agenti hanno la *stessa* conoscenza delle strategie ed inoltre questa è *completa*.

Si ipotizza che questa sia la configurazione che presenterà le migliori prestazioni complessive.

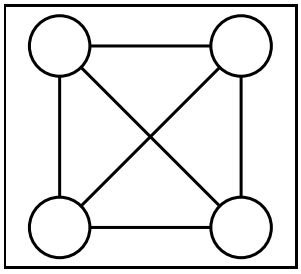

Figura 4.6: Connessione completa degli agenti.

Figura 4.7: Finestra per la gestione della simulazione.

4.10 Interfaccia utente

In questo paragrafo viene descritta l'interfaccia con cui viene gestita la simulazione.

4.10.1 La barra principale

La barra principale in fig.4.7 presenta i bottoni

- *Load Model*: carica una simulazione.
- *Play*: fa partire una simulazione.
- *Step*: fa girare la simulazione un passo alla volta.
- *Initialize*: inizializza la simulazione.
- *Stop*: ferma la simulazione.

- *Pause*: mette in pausa la simulazione.
- *Setup Model*: permette di riconfigurare la simulazione, modificando i parametri.
- *View Parameters*: mostra i parametri della simulazione.
- *Exit*: esce dalla piattaforma.

4.10.2 La gestione dei parametri

I parametri della simulazione vengono gestiti nella finestra di fig.4.8 e i campi hanno il seguente significato:

- *Agent*: numero dell'agente per cui dev'essere generato il diagramma dello stato.
- *Amplitudes*: tipo di strategia ad onda quadra.
- *BaseAmp*: ampiezza base della strategia ad onda quadra.(valore usato solo se amplitudes = Random)
- *Connection*: tipologia di connessione dei giocatori.
- *Epsilon*: grado si sperimentazione degli agenti.
- *Gamma*: grado di velocità di apprendimento. Un valore alto implica maggiore velocità, ma minore accuratezza.
- H: valore h della matrice di payoff del gioco.
- K: valore k della matrice di payoff del gioco.
- *Memory*: se la checkbox è spuntata la domanda ha una distribuzione con memoria.

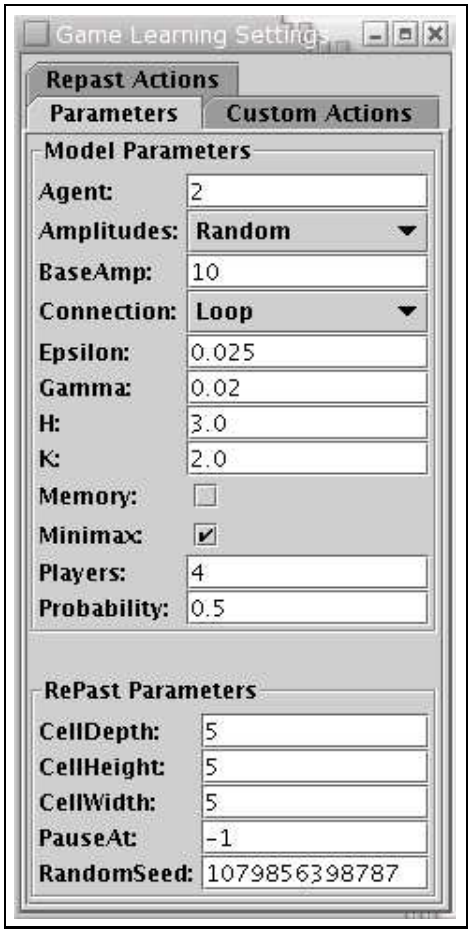

Figura 4.8: Finestra per la gestione dei parametri.

- *Minimax*: usa la strategia minimax anziché la strategia mista.
- *Players*: numero dei giocatori.
- *Probability*: probabilità della strategia *compro*(usata se la strategia dell'agente **Chance** e di tipo *senza memoria.*)

4.10.3 Grafici in uscita

La simulazione dà in uscita quattro tipi di grafici ognuno dei quali rende conto di vari aspetti della simulazione.

Grafico del costo medio In fig.4.9 viene riportato l'andamento di un costo medio dove in ascissa c'è il tempo di simulazione che coincide col numero di partite giocate mentre in ordinata c'è il valore del costo medio.

Questa è l'uscita di principale interesse dato che l'obiettivo delle decisioni è quello di minimizzare i costi delle scorte di magazzino.

Grafico dello stato medio In fig.4.10 viene riportato un andamento dello stato medio dove in ascissa c'è il tempo di simulazione, che coincide col numero di partite giocate, mentre in ordinata c'è il valore dello stato medio ottenuto ponendo, per ogni giocatore lo stato *produco* pari a 1 e lo stato *non produco* pari a 0 (i.e., un valore di $0,25$ indica che $\frac{1}{4}$ è nello stato *produco* mentre la restante parte è nello stato non *produco*). In ascissa oltre al tempo è mostrato il valore di $\lambda_2(L)$ denotato con *2EigL* per la connessione scelta.

Grafico dello stato di un agente In fig.4.11 viene riportato un andamento dello stato di un agente dove in ascissa c'è il tempo di simulazione, che coincide col numero di partite giocate, mentre in ordinata c'è il valore

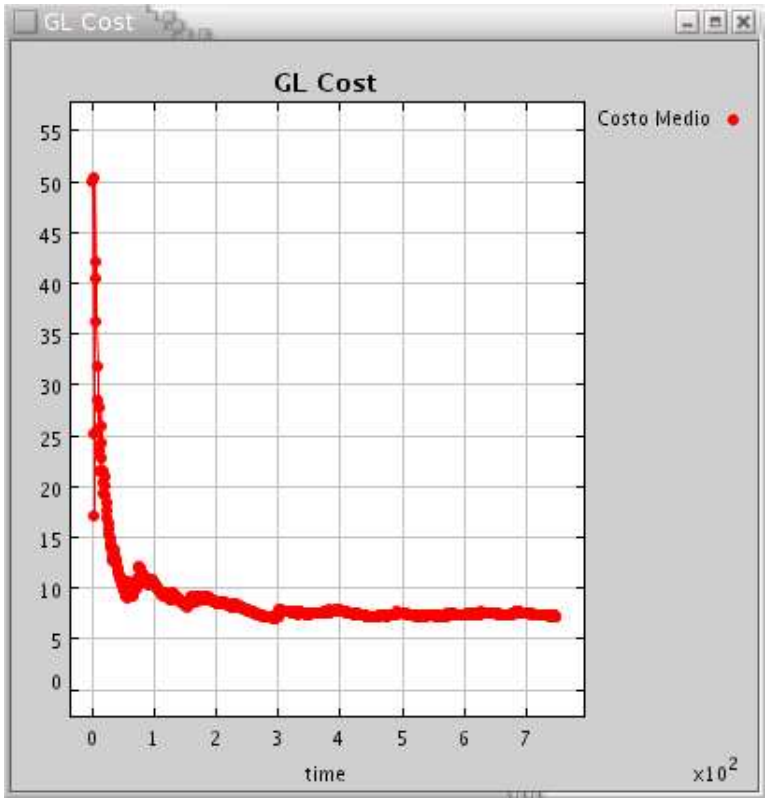

Figura 4.9: Grafico del costo medio.

dello stato ottenuto ponendo, per ogni giocatore lo stato *produco* pari a 1 e lo stato *non produco* pari a 0.

La scelta di gestire questa uscita è motivata dalla volontà di poter osservare l'andamento dello stato di un agente dato che l'andamento delle scelte strategiche è determinabile facilmente dall'andamento dello stato medio solo quando quest'ultimo ha basse oscillazioni.

Display degli agenti In fig.4.12 viene riportato lo schermo su cui vengono disegnati gli ovali che rappresentano gli agenti. Dentro la figura ovale viene scritto lo stato all'istante t e vengono anche disegnate le connessioni fra gli agenti.

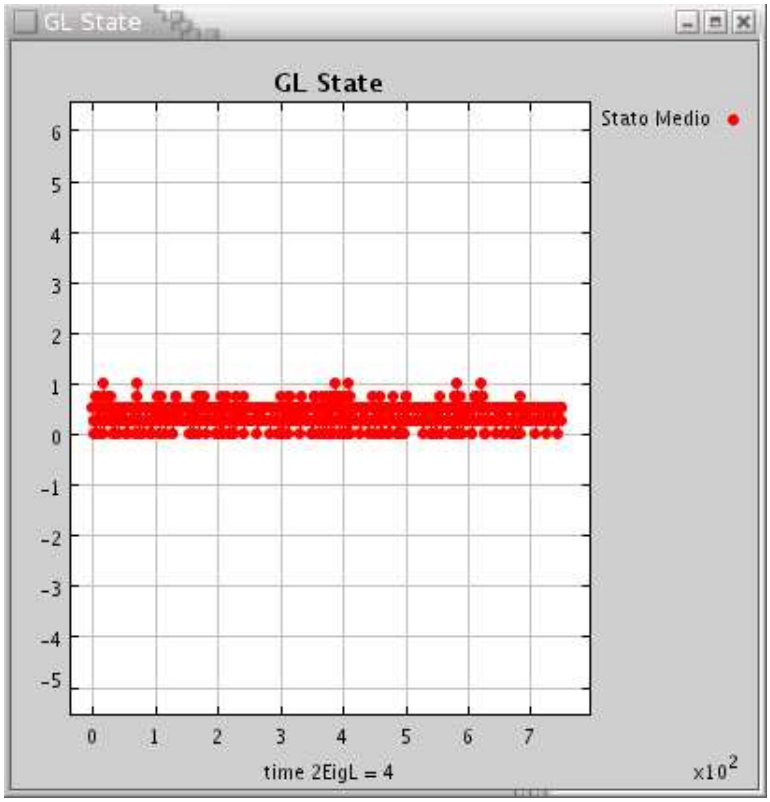

Figura 4.10: Grafico dello stato medio.

Lo stato viene denotato in questo modo:

- Per l'agente col ruolo *Agent*: **NP** denota lo stato *non produco* mentre **P** denota lo stato *produco*.
- Per l'agente col ruolo *Chance*: **NC** denota lo stato *non compro* mentre **C** denota lo stato *compro*.

Per motivi dipendenti dalla libreria per la visualizzazione il display può essere molto difficoltoso da seguire se il numero di agenti è elevato (più di 100 agenti di tipo *Agent*) poiché la figura degli agenti è molto piccola.

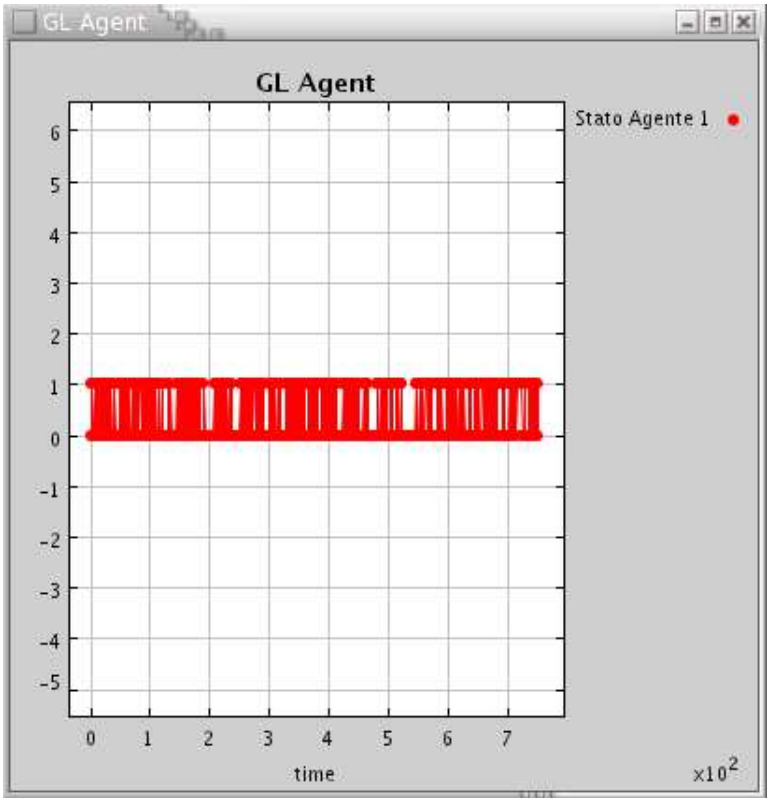

Figura 4.11: Grafico dello stato di un agente.

Nota Le connessioni non vengono visualizzate in due casi:

- Non si è scelto di usare la strategia mista nei parametri della simulazione. Gli agenti, usando la strategia mista, non utilizzano la connessione.
- La connessione è completa ed il numero di agenti è superiore a 16. La connessione non è visualizzata poiché l'alto numero di archi oscura gli agenti.

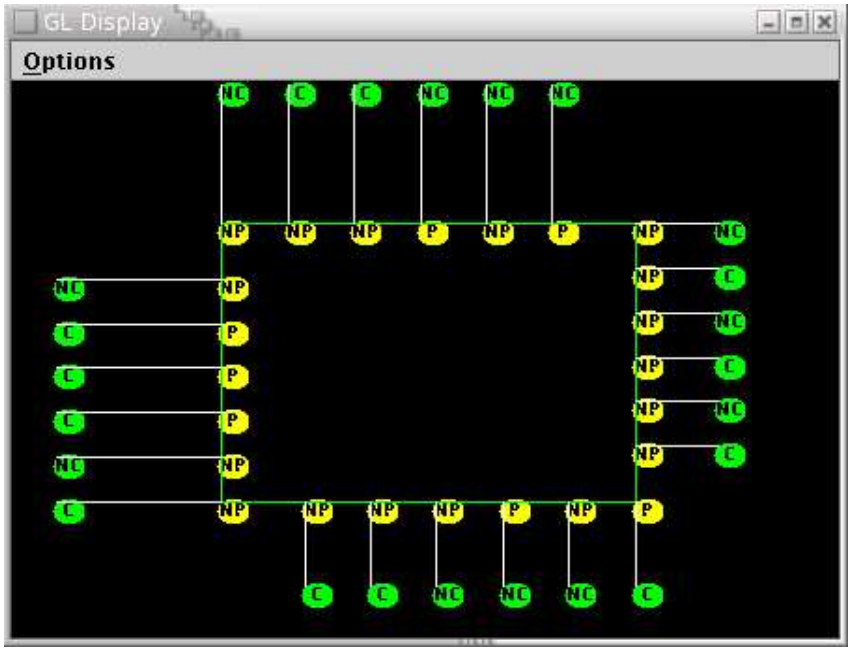

Figura 4.12: Display degli agenti.

4.11 Note sulla generazione dei numeri casuali

Dal file Random.java[5] si osserva che la costruzione del generatore di Mersenne Twister viene effettuata in modo unico con questo costruttore:

```
static {
   Date d = new Date();
   rngSeed = d.getTime();
   generator = new MersenneTwister(d);
}
```

e tutte le distribuzioni generate dalla stessa istanza della classe **Random** hanno un unico generatore di base.

[5]Path relativo src/uchicago/src/sim/util/

Capitolo 5

Simulazione: risultati

In questo capitolo vengono esposte le simulazioni sulle reti di agenti e vengono riportati i risultati.

5.1 Condizioni stazionarie

Allo scopo di testare le capacità di apprendimento dell'algoritmo RLA si sono fatti alcuni test in cui l'uscita dell'agente **Chance** era costante e l'agente **Agent** utilizzava una strategia mista slegata dall'FSA.

Le simulazioni sono state fatte nelle seguenti condizioni: $\gamma = 0.02$, $\epsilon = 0.025$, $h = 3.0$, $k = 2.0$, $\left.\frac{k}{n(t)}\right|_{n(t)=0} = 200$

Nel caso in cui l'agente **Chance** usava la strategia *non compro* con $p = 0.005$ l'agente **Agent** giocava la strategia pura *non produco* e ricavava come payoff delle strategie:

- *non produco*: 0 com probabilità: 0.975.
- *produco*: 250 con probabilità: 0.025

l'andamento del costo medio è riportato in fig. 5.1.

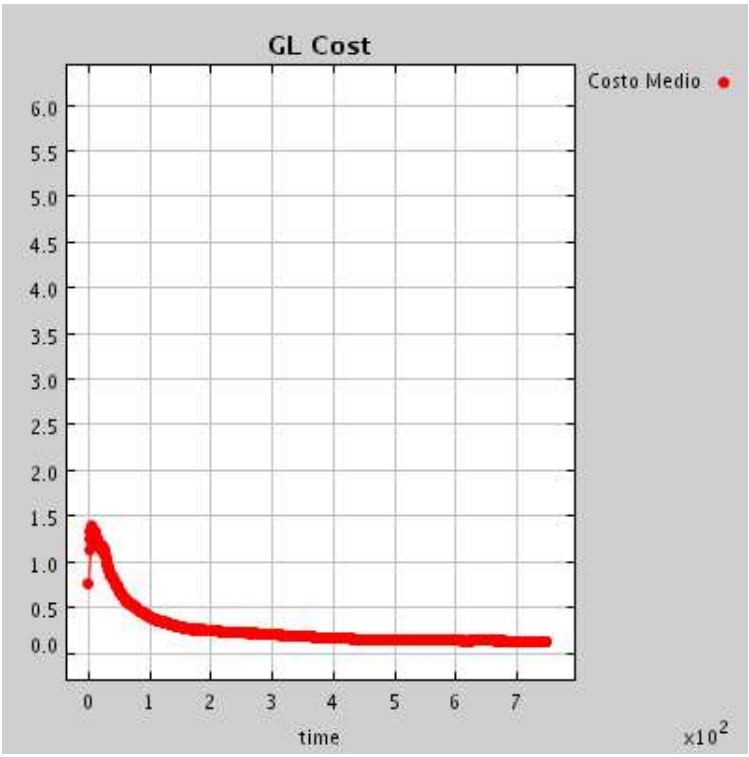

Figura 5.1: Andamento del costo medio (Strategia **Chance**=*non compro*)

Nel caso in cui l'agente **Chance** usava la strategia *compro* con $p = 0.995$ l'agente **Agent** giocava la strategia pura *produco* e ricavava come payoff delle strategie:

- *non produco*: 3 con probabilità: 0.025.
- *produco*: 0 con probabilità: 0.975

l'andamento del costo medio è riportato in fig.5.2

Da questi diagrammi si osserva che il costo medio tende a zero com'era presumibile dato che uno dei requisiti di base dell'algoritmo è che in queste condizioni la strategia scelta dev'essere quella *non dominata*.

Utilizzando l'agente **Agent** con il vincolo sulla decisione di non produrre

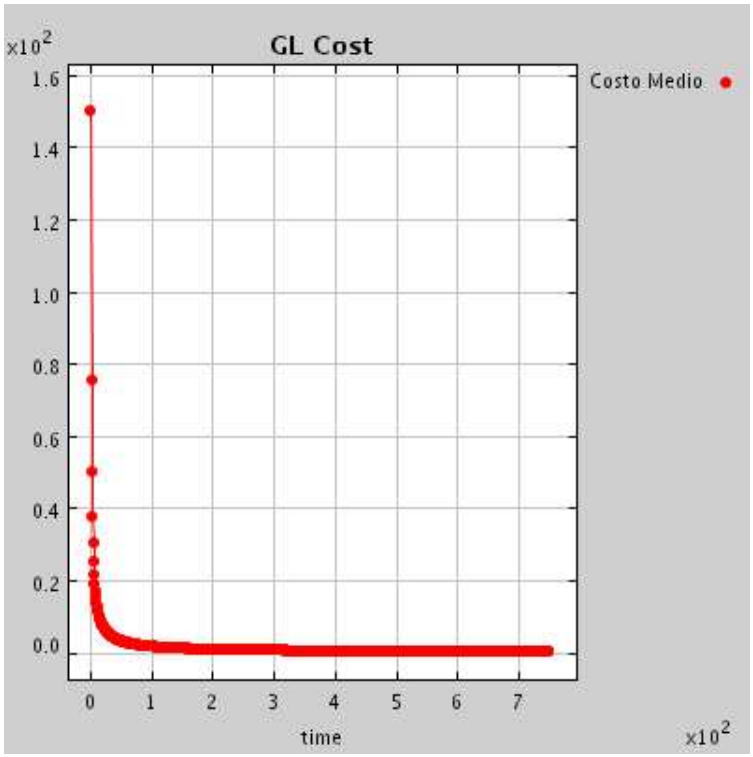

Figura 5.2: Andamento del costo medio (Strategia **Chance**=*compro*).

si osserva che poiché un dato del problema è che l'agente **Chance** è allo stato *compro*, l'agente **Agent** viene vincolato a trovarsi nello stato *non produco* senza utilizzare la strategia mista, ottenendo che il costo medio tende al valore $\frac{k}{n(t)}$ che non è gestibile (cfr 4.4).

L'andamento del costo medio è riportato in fig.5.3.

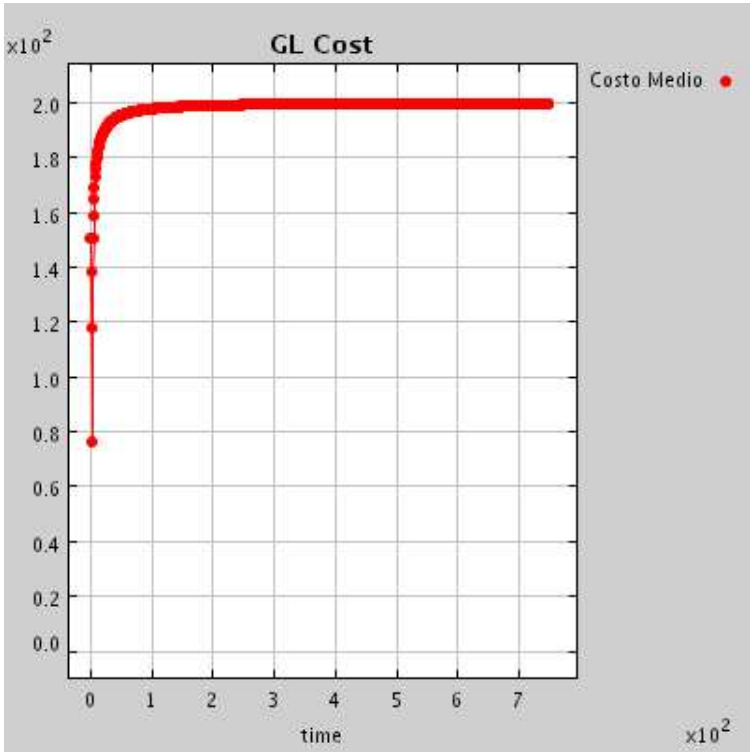

Figura 5.3: Andamento del costo medio col vincolo sul comportamento.

5.2 Condizioni non stazionarie

5.2.1 Impatto della strategia mista

Al fine di valutare l'impatto della coordinazione e della scelta di utilizzare, al posto della strategia mista, un criterio di tipo *minimax* sono state effettuate delle simulazioni.

Per effettuare i confronti si è scelto, arbitrariamente, di fissare il numero di agenti a 8 e di fissare il numero di partite complessivamente a 750.

Le simulazioni, tranne dove viene specificato altrimenti, sono state eseguite nelle seguenti condizioni: $\gamma = 0.02$, $\epsilon = 0.025$, $h = 3.0$, $k = 2.0$, $\left.\frac{k}{n(t)}\right|_{n(t)=0} = 200$.

Il giocatore **Chance** gioca una strategia *senza memoria* con $p = 0.5$.

In fig.5.4 è mostrato l'andamento del costo medio, per la strategia mista, che tende ad un valore prossimo a 90 ed, inoltre, si osserva che l'andamento è di tipo esponenziale.

In fig.5.5 è mostrato, invece, l'andamento dello stato medio che ha un'andamento prossimo al valore di *non produco* esplorando l'altra strategia sporadicamente. La scarsa presenza del valore 1 si giustifica tenendo conto del vincolo sul comportamento e considerando che $k < h$.

In altri esperimenti si vedrà come una maggiore differenza tra i valori di k e di h migliori il comportamento.

Per valutare le prestazioni della strategia mista si sceglie come connessione quella ad anello poiché si suppone che sia la strategia dalle prestazioni a metà fra quella migliore e quella peggiore.

In fig.5.6 è mostrato l'andamento del costo medio, con la strategia minimax, che tende ad un valore circa pari a 10, migliore del caso preceden-

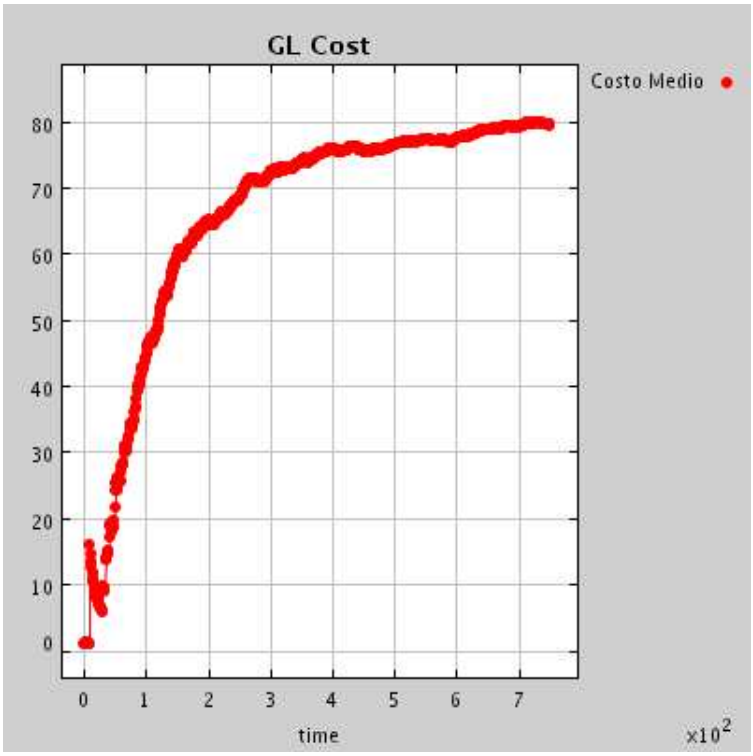

Figura 5.4: Andamento del costo medio (pura strategia mista).

te. Inoltre si osserva che l'andamento è di tipo oscillatorio rispetto a quello mostrato in fig.5.4.

In fig.5.7 è mostrato l'andamento dello stato medio che oscilla in un intervallo di valore più ampio rispetto al caso in cui viene utilizzata la strategia mista.

Si osserva come il costo medio tende ad un valore più basso per l'utilizzo delle informazioni ottenute dagli altri agenti che rende possibile l'uso di una strategia minimax.

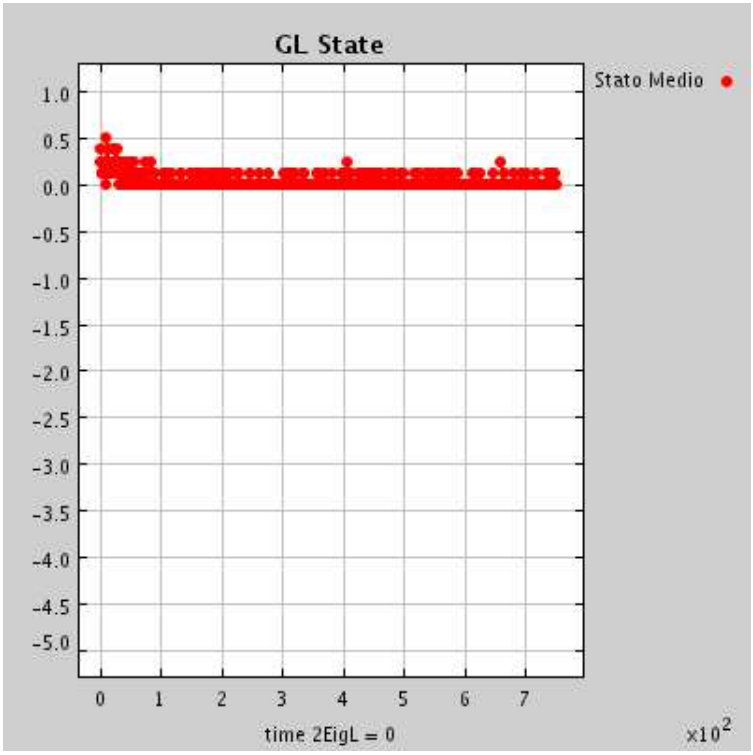

Figura 5.5: Andamento dello stato medio (pura strategia mista).

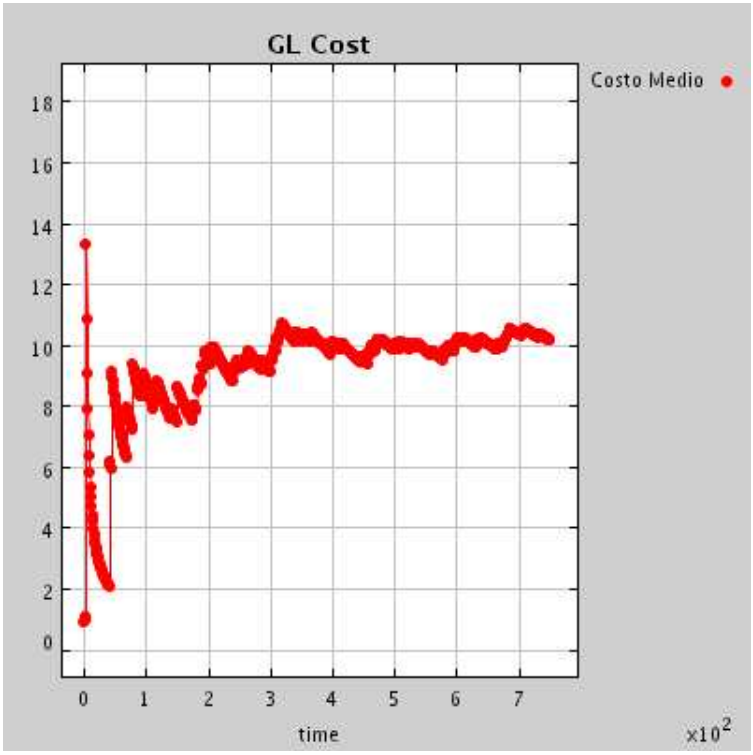

Figura 5.6: Andamento del costo medio (strategia minimax).

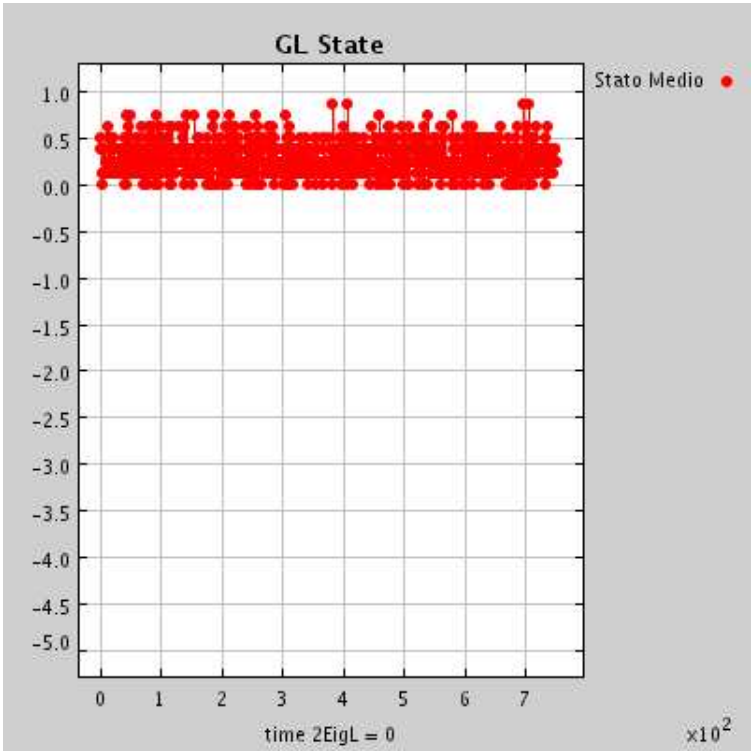

Figura 5.7: Andamento dello stato medio (strategia minimax).

5.3 Connessioni

Di seguito si analizzano i risultati relativi alle singole connessioni al fine di valutare, ed evidenziare quando possibile, le differenze prestazionali nei vari casi.

La strategia mista non è presa in considerazione in queste simulazioni dato che la scelta non è basata sui messaggi provenienti dagli altri agenti.

5.3.1 Domanda *senza memoria*

Per effettuare i confronti si è scelto, arbitrariamente, di fissare il numero di agenti a 8 e di fissare il numero di partite complessivamente a 750.

Le simulazioni, tranne dove viene specificato altrimenti, sono state fatte nelle seguenti condizioni: $\gamma = 0.02$, $\epsilon = 0.025$, $h = 3.0$, $k = 2.0$, $\left.\frac{k}{n(t)}\right|_{n(t)=0} = 200$.

Il giocatore **Chance** gioca una strategia *senza memoria* con $p = 0.5$.

Connessione a catena Dal grafico del costo medio, rappresentato in fig.5.8, si osserva che il suo valore finale tende a un valore inferiore a 10 e che il suo andamento presenta caratteri di oscillazione.

Connessione ad anello Il grafico del costo medio è omesso in quanto già discusso e rappresentato in fig.5.6.

Connessione a stella Dal grafico del costo medio, rappresentato in fig.5.9, si osserva che il suo valore finale tende ad un valore più basso rispetto ai casi in cui la connessione è a catena o ad anello, visto che il valore finale tende a sei.

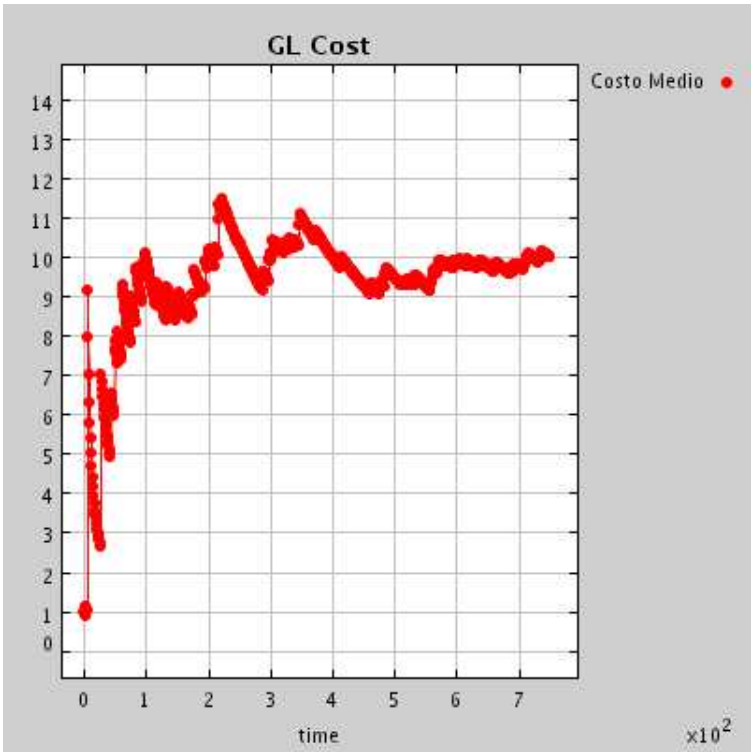

Figura 5.8: Andamento del costo medio (connessione a catena).

Connessione completa Dal grafico del costo medio, rappresentato in fig.5.10, si osserva che il suo valore finale è superiore a dieci anche se l'andamento dello stato, mostrato in fig.5.11 oscilla in un intervallo di valori con un ampiezza limitata.

La bassa oscillazione può essere spiegata con la considerazione che ogni nodo della rete ha un'informazione completa del comportamento degli altri agenti all'istante $t-1$, comportando una minore variazione della stima $\tilde{N}$ in giochi consecutivi.

Dall'analisi di queste esperimenti si evince che nel caso in cui la domanda è *senza memoria* gli andamenti dei costi medi sono abbastanza simili per tutte le connessioni prese in esame, tranne nel caso in cui la connessione è

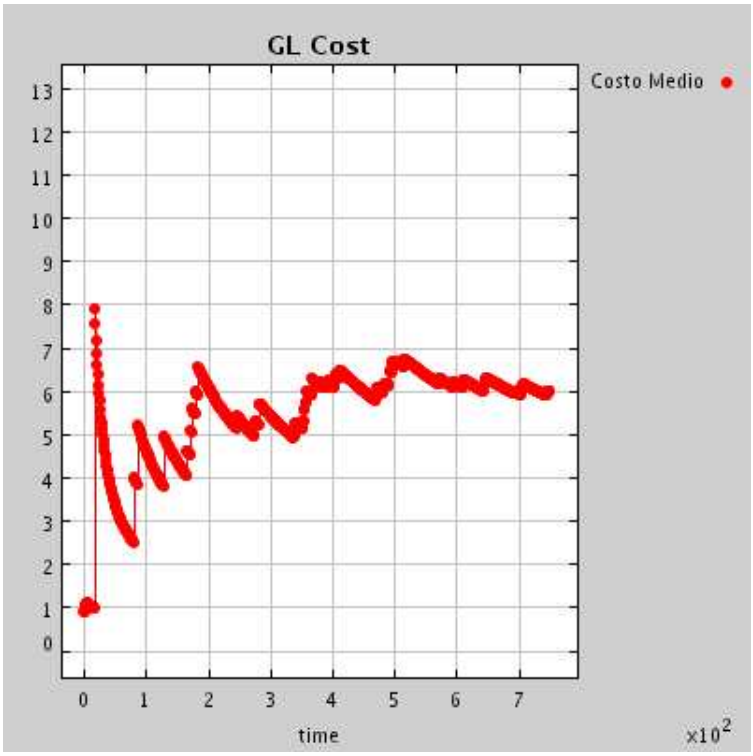

Figura 5.9: Andamento del costo medio (connessione a stella).

completa in cui l'andamento è più simile, a meno del valore finale al caso in cui gli agenti usano la strategia mista.

Dato che l'andamento degli stati per la connessione ad anello e per la connessione a stella è sostanzialmente uguale all'andamento di fig.5.7 questi andamenti vengono omessi per brevità.

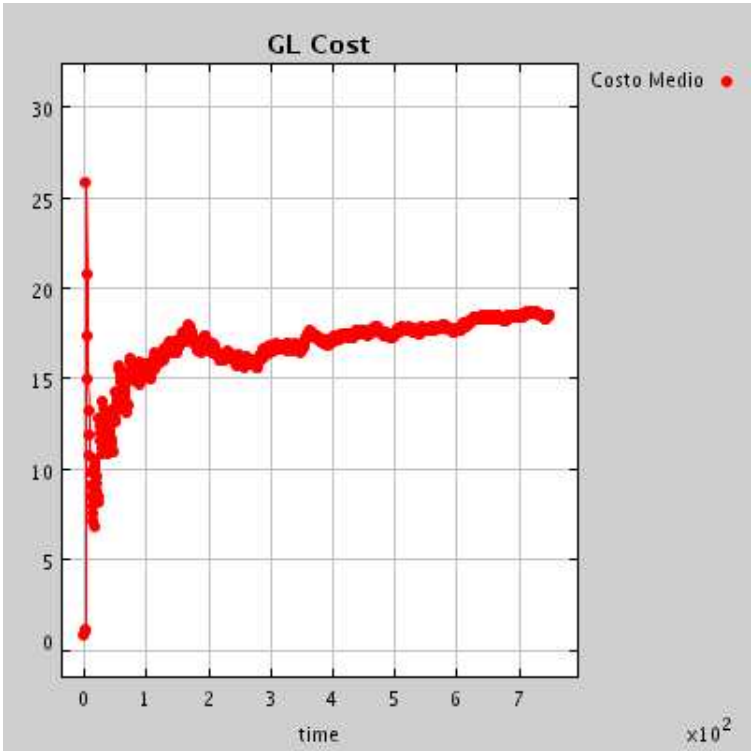

Figura 5.10: Andamento del costo medio (connessione completa).

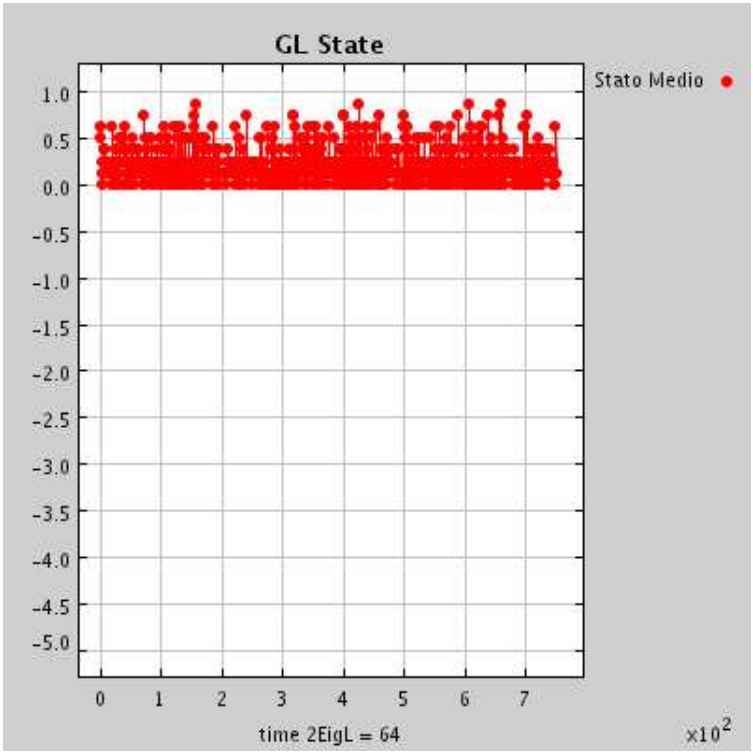

Figura 5.11: Andamento dello stato medio (connessione completa).

5.3.2 Domanda con memoria

Per valutare le differenze nel caso in cui l'agente di tipo **Chance** utilizzi una strategia modellabile con una distribuzione di probabilità con memoria sono state effettuate le simulazioni e ne vengono riportati i risultati.

Per effettuare i confronti si è scelto, arbitrariamente, di fissare il numero di agenti a 8 e di fissare il numero di partite complessivamente a 750.

Le simulazioni, tranne dove viene specificato altrimenti, sono state fatte nelle seguenti condizioni: $\gamma = 0.02$, $\epsilon = 0.025$, $h = 3.0$, $k = 2.0$, $\left.\frac{k}{n(t)}\right|_{n(t)=0} = 200$.

Connessione ad anello Dato che le prestazioni di queste connessioni sono state le migliori nel caso di domanda *senza memoria* viene utilizzata come prima configurazione nella simulazione.

In fig.5.12 è riportato l'andamento del costo medio da cui si osserva che è peggiore del caso 5.6 trattato nel precedente paragrafo presentando oscillazioni più evidenti ed un valore finale più elevato.

In fig.5.13 è riportato l'andamento dello stato medio da cui si osserva come lo stato medio assuma un valore inferiore a 0.5 per quasi tutte le partite effettuate.

Connessione completa In fig.5.14 è riportato l'andamento del costo medio nel caso in cui la connessione sia completa. Come nel caso precedente, le prestazioni sono peggiori nel caso in cui la domanda abbia una distribuzione di probabilità *senza memoria* anche se con oscillazioni meno evidenti.

Si osserva come il valore finale sia più alto rispetto alla connessione ad anello, e con un andamento quasi esponenziale.

In fig5.15 è riportato invece l'andamento dello stato medio dove si osserva

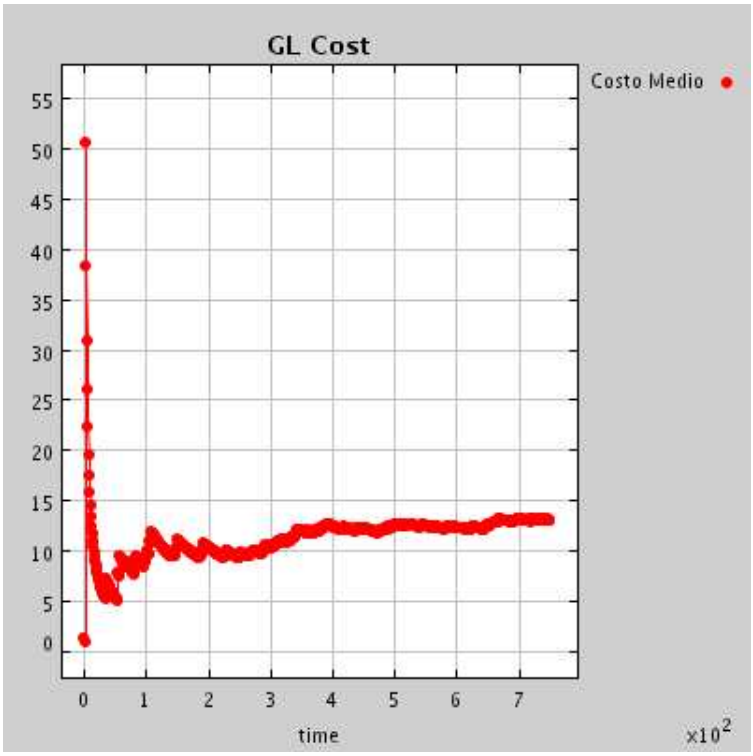

Figura 5.12: Andamento del costo medio (connessione ad anello).

un'oscillazione più stretta rispetto alla connessione ad anello, dato che nella maggior parte delle partite sia inferiore al valore 0.5.

In esperimenti successivi si osserverà che l'ampiezza delle oscillazioni dello stato è inversamente proporzionale al grado di connessione.

Per vedere se l'aumento del numero di agenti avesse un'impatto si è scelto di raddoppiare il numero di agenti ottenendo il grafico di fig.5.16 nel quale si osserva che il valore finale è più alto di quello rappresentato in fig.5.14 e con un andamento meno oscillatorio.

Inoltre, dall'andamento dello stato medio in fig.5.17, si vede come questo abbia un oscillazione più stretta osservando che in poche partite lo stato medio ha un valore maggiore di 0.4.

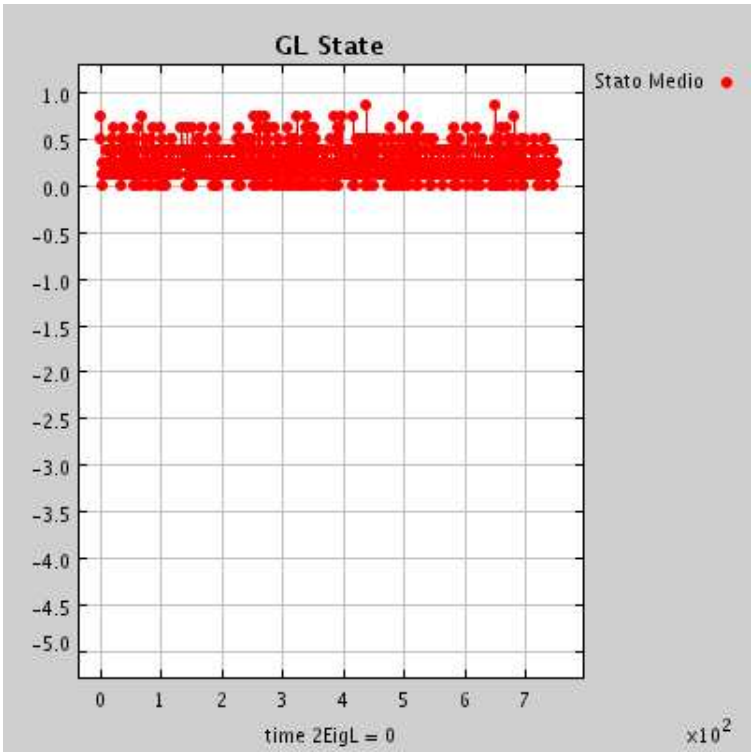

Figura 5.13: Andamento dello stato medio (connessione ad anello).

A scopo di confronto, è stato fatto un esperimento utilizzando la strategia mista ottenendo l'andamento in fig.5.18 che è peggiore di qualunque connessione che usa la strategia minimax mentre l'andamento dello stato medio è simile a quello già mostrato in fig.5.5.

Inoltre in fig.5.19 è mostrato l'andamento della connessione ad anello con 16 agenti dove si osserva come sia migliore del caso con 8 agenti, mentre l'andamento dello stato medio essendo simile a quello di fig.5.6 viene omesso per brevità.

Questo conferma l'ipotesi che il comportamento complessivo degli agenti migliori all'aumentare del grado di connessione ma rimane poco chiaro perché l'andamento della connessione completa sia peggiore di quella ad anello,

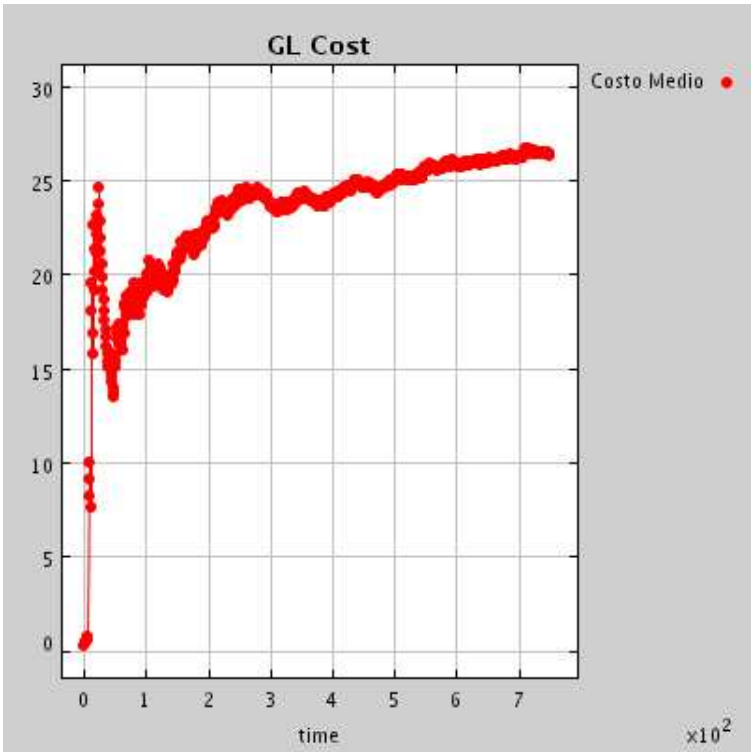

Figura 5.14: Andamento del costo medio (connessione completa) 8 Agenti.

anche se si osserva che c'è una certa somiglianza tra gli andamenti degli stati medi tra la connessione completa e la configurazione in cui viene usata la strategia mista.

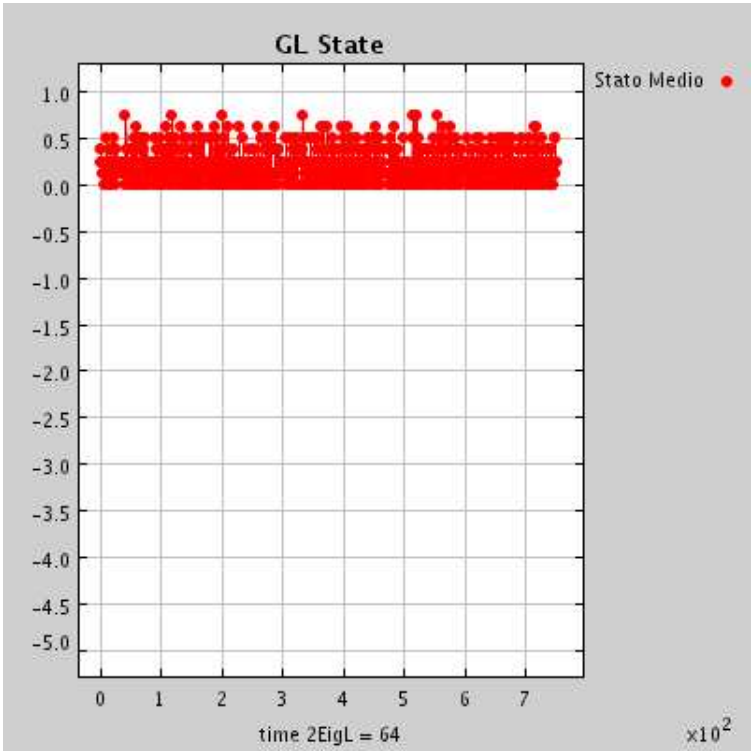

Figura 5.15: Andamento dello stato medio (connessione completa) 8 Agenti.

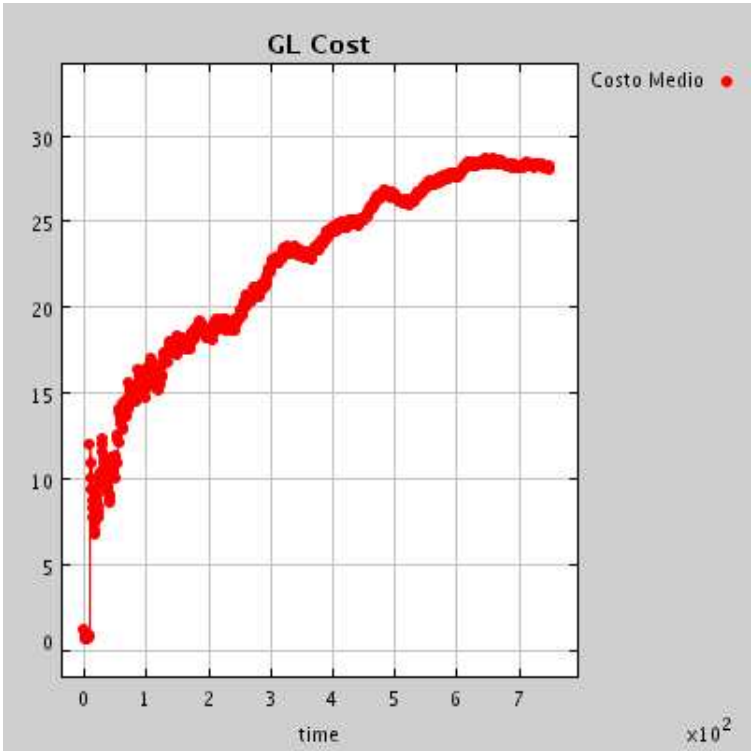

Figura 5.16: Andamento del costo medio (connessione completa) 16 Agenti.

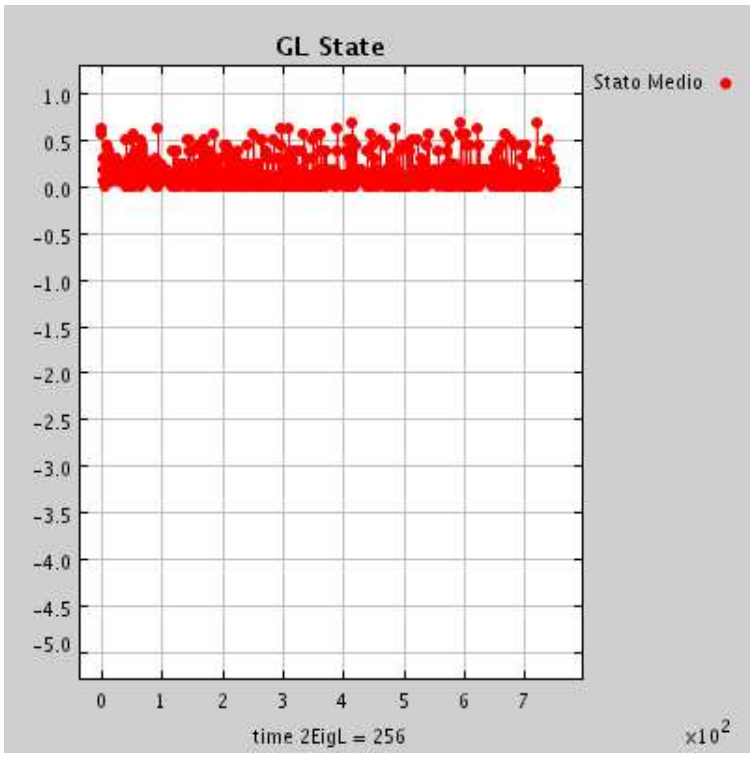

Figura 5.17: Andamento dello stato medio (connessione completa) 16 Agenti.

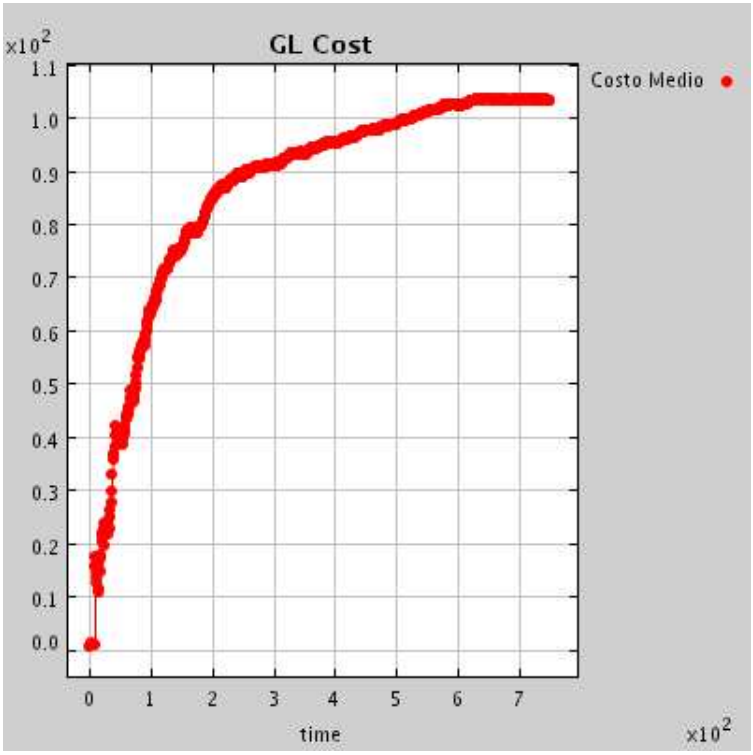

Figura 5.18: Andamento del costo medio (strategia mista) 8 Agenti.

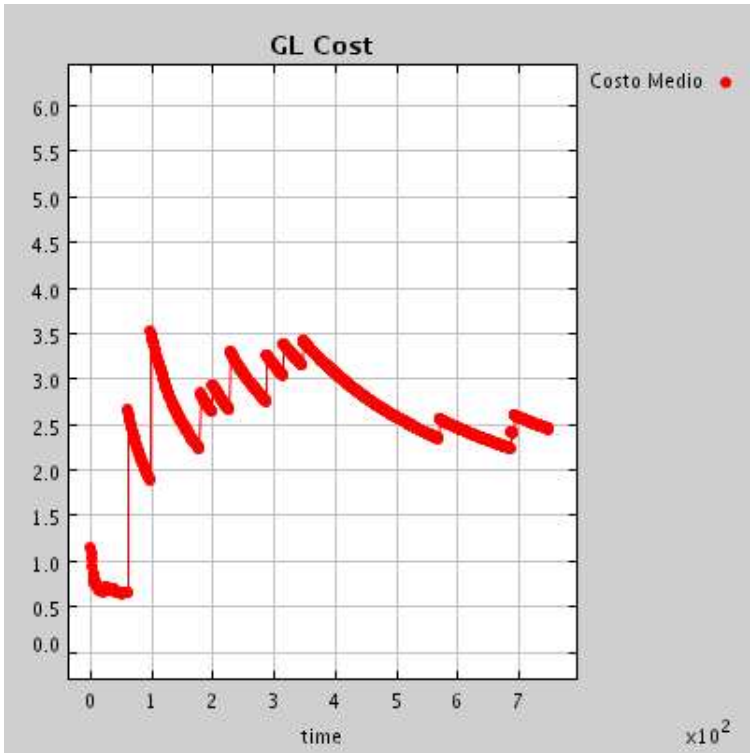

Figura 5.19: Andamento del costo medio (connessione ad anello) 16 Agenti.

5.4 Apprendimento e domanda

Nel caso in cui l'andamento della domanda sia del tipo mostrato in fig.5.20, è possibile vedere se il comportamento dell'apprendimento sia prevedibile.

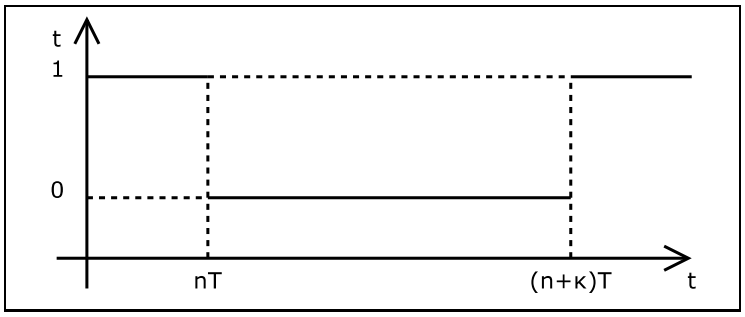

Figura 5.20: Andamento delle strategie di **Chance**.

Per un numero di partire pari a k l'avversario **Chance** gioca sempre la strategia *non compro*[1] mentre l'algoritmo di apprendimento aggiorna la strategia scelta secondo

$$p_i^{t+1} = p_i^t + \gamma \pi_i^t \sum_{j \neq i} a_j^t p_i^t$$

in queste condizioni si ha che π_i^t varierà secondo due valori di cui uno è zero.

Il giocatore gioca il gioco rappresentato in tab.5.1 che è risolubile per dominanza ed una delle proprietà dell'algoritmo è quella che in condizioni stazionarie gioca strategie non dominate.

In questa situazione si ha che, se denotiamo con $\tilde{p}_i^t$ la strategia dominante, il termine $\pi_i^t = \text{cost} = \tilde{\pi}_i^t$ determnina un fattore di aggiornamento che, tenuto conto che $a_j^t = \min\left[1, \dfrac{p_j^t - \epsilon/(N-1)}{\gamma \pi_i^t p_j^t}\right]$, fa sì che la probabilità al tempo $t+1$ possa essere scritta come:

[1]Non contemplando la situazione in 4.4 per l'altra strategia il ragionamento è analogo

Natura/Agente	**non produco**	**produco**
non compro	0	h

Tabella 5.1: Payoff del gioco ridotto.

$$\tilde{p}_i^{t+1} = p_i^t \Big($$

( ) 583 7 97 1 89 (( ) 583 7 97 7 97) 4 8 0 29 6 57 816 9552 0 110 1 0

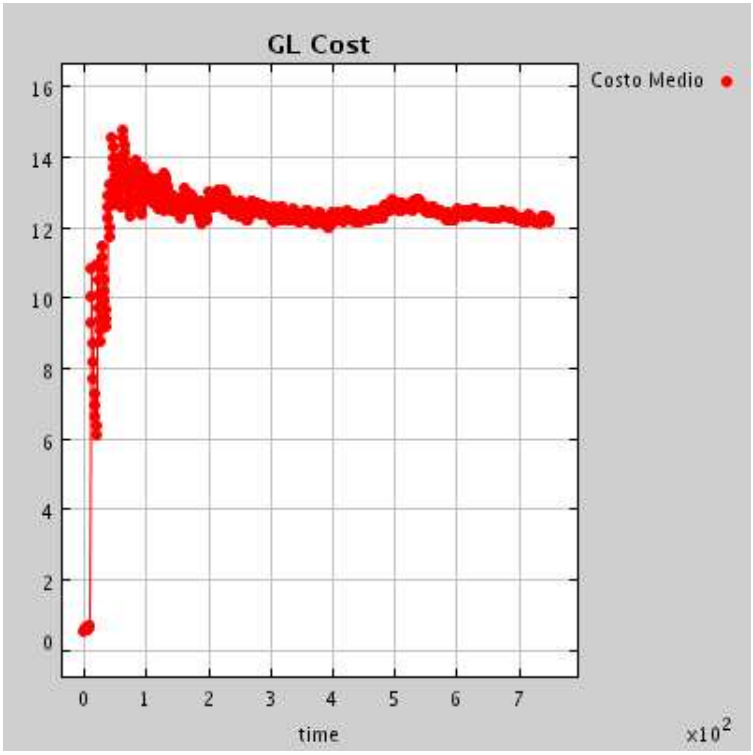

Figura 5.48: Andamento del costo medio (Connessione completa, $\gamma = 0.25$, 16 Agenti).

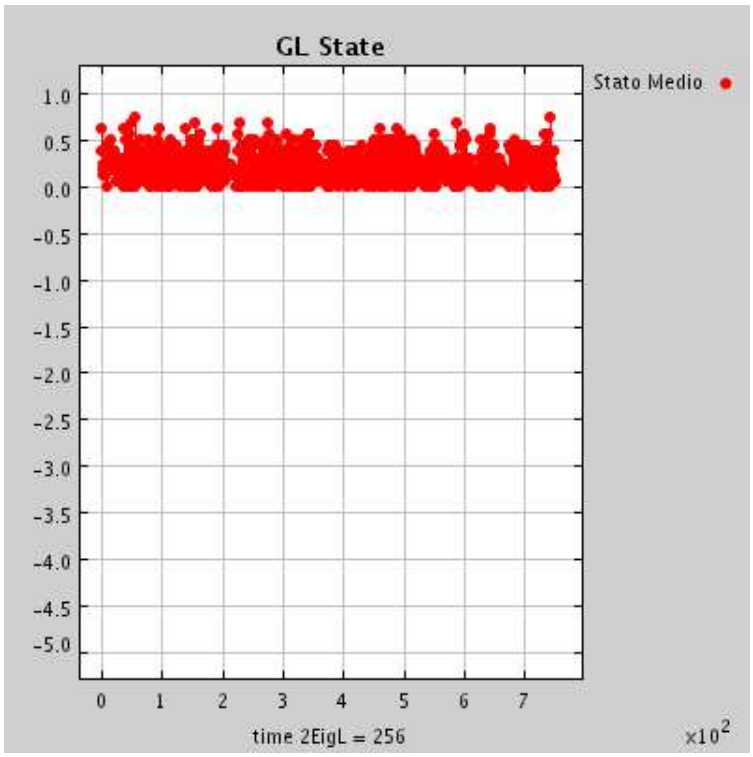

Figura 5.49: Andamento dello stato medio (Connessione completa, $\gamma = 0.25$, 16 Agenti).

5.7 Annotazioni

Scopo di questa sezione è quella di precisare alcuni aspetti dei grafici poco visibili a causa principalmente della scala.

Oscillazioni del costo Nei grafici del costo quelle che appaiono come oscillazioni in realtà sono delle ondulazioni tipo *dente di sega* come si può vedere in fig.5.50.

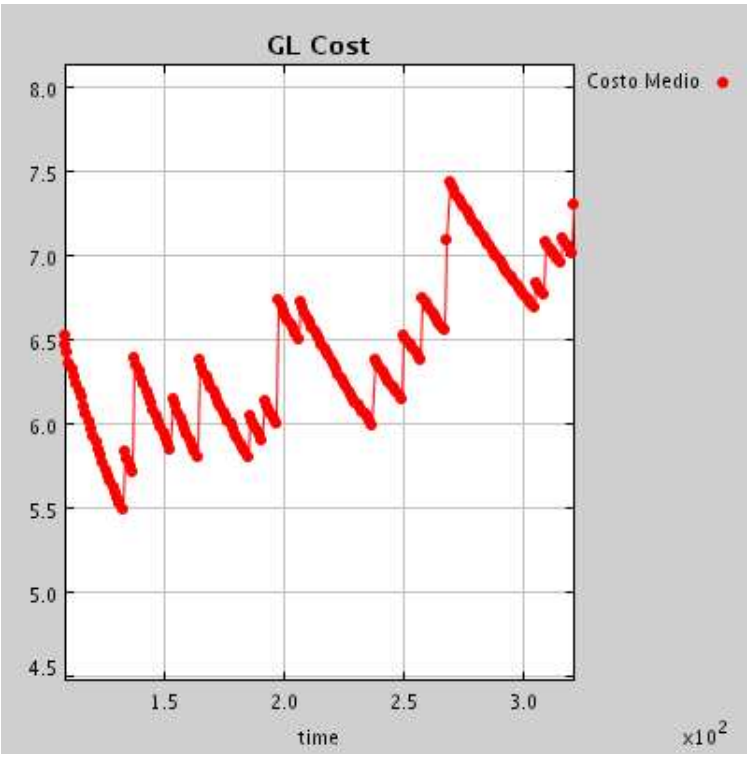

Figura 5.50: Andamento del costo medio (particolare).

Oscillazioni dello stato Nei grafici dello stato quelle che appaiono come bande sono oscillazioni come quelle che sono mostrate in fig.5.51, cioè sono oscillazioni su due valori centrali intervallate da dei picchi.

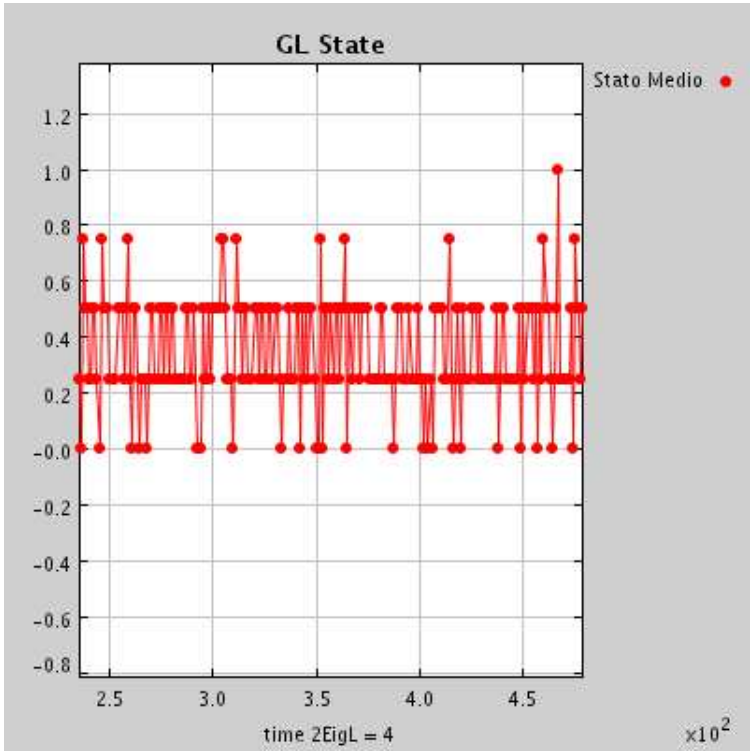

Figura 5.51: Andamento dello stato medio (particolare).

Capitolo 6

Conclusioni

In questo capitolo vengono analizzati i risultati delle simulazione e discussi i possibili miglioramenti futuri.

6.1 Principali risultati

Dalle simulazioni emerge come la scelta della strategia con un criterio *minimax* abbia prestazioni migliori di una strategia mista, però preme sottolineare che un criterio minimax può essere usato se il gioco è cooperativo oppure se la struttura della matrice dei payoff è nota, mentre la strategia mista ha come requisito la semplice conoscenza delle strategie e, quindi, può essere usata anche in contesti non cooperativi.

Le prestazioni degli agenti migliorano all'aumentare del grado di connessione e questo può essere spiegabile col miglioramento della stima dei costi variabili dovuto alla maggiore informazione che riceve l'agente.

Dall'analisi del $\lambda_2(L)$ si è osservato come all'aumentare di questo valore corrisponda un intervallo di convergenza più stretta e un comportamento meno oscillatorio dell'agente nella scelta della strategia.

Dalle simulazioni è emerso che, in queste condizioni, con la strategia minimax non è indispensabile avere una connessione completa per ottenere i migliori risultati in termini di costo medio, anche se la connessione completa ha evidenziato un comportamento poco dipendente dal tipo di strategia in ingresso. Altre strategie hanno evidenziato risultati, in alcuni casi, molto dipendenti dall'ingresso e dal numero di agenti.

Lo stato medio risulta in quasi tutti i casi di tipo oscillatorio, e si è osservato che converge in maniera evidente quando l'ingresso è di tipo stazionario o quasi stazionario.

6.2 Convergenza delle decisioni

Dalle prove emerge come una convergenza eccessiva delle decisioni, legata ad un valore di $\lambda_2(L)$ elevato o a un basso valore di ϵ, porti a dei risultati peggiori rispetto a decisioni con andamento oscillante.

Il valore del protocollo di consenso, che spinge lo stato degli agenti a convergere allo stato medio, dipende da $\lambda_2(L)$ poiché si è osservato, in sede di modello, che all'aumentare della connessione aumenta la precisione della stima di $N(t)$. Poiché $\tilde{n}(t) = u_i(t) - \frac{1}{N}x_i(t-\tau)$ all'aumentare di N questo valore fa si che la stima del costo variabile $\tilde{\pi}_i$ sia più bassa rispetto ad una configurazione con un basso valore di $\lambda_2(L)$. I costi associati dipendono maggiormente dalle probabilità risultanti dall'algoritmo RLA.

Le prove mostrano come quest'algoritmo, dato che converge a D^∞, ha le migliori prestazioni se l'ingresso ha caratteri di stazionarietà poiché l'insieme D^∞ in questo caso non è vuoto oppure se c'è una grossa differenza numerica fra i parametri k ed h del gioco. Infatti nei casi in cui la domanda ha maggiori caratteri di oscillazione è risultata più marcata la differenza nei risultati tra

la strategia minimax e la strategia pura.

6.3 Miglioramenti

L'agente, nelle simulazioni svolte, usa o la strategia mista oppure la strategia minimax, non essendo gestita la possibilità che l'agente possa scegliere di usare entrambe le strategie, un possibile miglioramento del comportamento dell'agente potrebbe essere la scelta delle azioni da compiere usando una delle due strategie in base ad alcune misure sull'ambiente.

Le simulazioni sono state condotte per fissato numero d'agenti e connessione. Sarebbe interessante considerare il comportamento della rete se questi parametri fossero variabili nel tempo (i.e., supportando l'inserimento e l'eliminazione di un agente e la modifica della connessione *a run-time.*)

La simulazione non gestisce altri giochi che non siano l'*inventory*, algoritmi d'apprendimento diversi dall'*RLA*, connessioni definibili dall'utente e modifiche della connessione a run-time. Da questo punto di vista il software potrebbe essere migliorato gestendo giochi e connessioni definibili dall'utente.

Appendice A

Requisiti ed Analisi

A.1 Metodo di progetto

Il metodo di progetto utilizzato come guida è *Gaia* dato che, a differenza di altri metodi noti a chi scrive[1], è l'unico adattabile alla piattaforma scelta non avendo una piattaforma di riferimento e/o non specificando tipi particolari di agenti. Il metodo prevede che, dopo l'analisi dei casi d'uso, venga specificato il comportamento dell'agente e le modalità di comunicazione degli agenti.

A.2 Analisi dei casi d'uso

A.2.1 Attori

Gli attori sono descritti in tab.A.1.

[1]Prometheus, PASSI

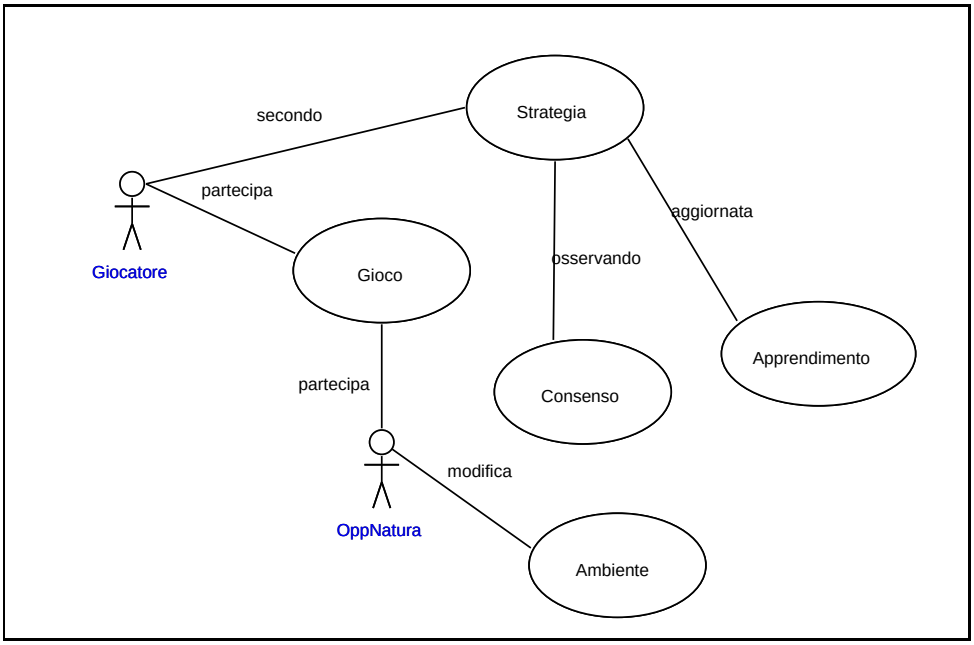

Figura A.1: Diagramma dei casi d'uso.

A.2.2 Casi d'uso

I casi d'uso che emergono dai requisiti sono descritti nelle tabelle da tab.A.2 a tab.A.6.

Giocatore	L'agente principale del sistema che modella un decisore che partecipa ad un gioco.
OppNatura	L'agente sistema che modella un decisore che partecipa ad un gioco secondo non una strategia ma secondo un modello aleatorio.

Tabella A.1: Attori.

Nome	Gioco
Attori coinvolti	Giocatore, OppNatura
Flusso d'eventi	1. Il giocatore sceglie la strategia. 2. L'OppNature sceglie la sua strategia. 3. Al giocatore viene comunicato il payoff ottenuto.

Tabella A.2: Caso d'uso Gioco.

Nome	Apprendimento
Attori coinvolti	Giocatore
Flusso d'eventi	1. Il giocatore considera il payoff ottenuto. 2. In base a questo aggiorna le probabilità in base all'algoritmo di apprendimento.

Tabella A.3: Caso d'uso Apprendimento.

Nome	Consenso
Attori coinvolti	Giocatore
Flusso d'eventi	1. Il giocatore osserva la scelta dei concorrenti adiacenti (gli altri giocatori). 2. Usa i valori osservati come elemento per la sua scelta.

Tabella A.4: Caso d'uso Consenso.

Nome	Strategia
Attori coinvolti	Giocatore
Flusso d'eventi	1. Per scegliere l'azione da fare il giocatore considera. 1.1 Le probabilità attuali. 1.2 Le azioni dei giocatori connessi. 1.3 I payoff che conosce al tempo t. 2. Il giocatore ottiene i dati per sceglie l'azione da fare.

Tabella A.5: Caso d'uso Strategia.

Nome	Ambiente
Attori coinvolti	Giocatore, OppNatura
Flusso d'eventi	1. Dopo un certo tempo (Timer o contatore?) modifica l'ambiente. 2. L'algoritmo d'apprendimento deve fare le sue modifiche.

Tabella A.6: Caso d'uso Ambiente.

Appendice B

Modello degli Agenti

Come viene trattato in [Wool00] un modello ad agenti è completo se vengono definiti i ruoli dell'Agente (*Modello dell'Agente*) e le interazioni tra i vari tipi di Agenti (*Società di Agenti*).

B.1 Ruoli

I ruoli che emergono dall'analisi dei casi d'uso sono due: Giocatore e Natura che sono competenza di due tipi d'Agenti diversi poichè sono molto diverse le loro proprietà e sono descritti in tab.B.1 e tab.B.2.

B.2 Interazioni

Ci sono due tipi di interazione: l'interazione tra due Giocatori e l'interazione tra Giocatore e Natura. Le interazioni sono descritte in tab.B.3 e in tab.B.4.

Ruolo	Giocatore
Descrizione	Modella il giocatore che deve prendere le sue decisioni.
Permessi	Può conoscere il payoff associato alle sue scelte ma non la struttura del gioco. Può conoscere lo stato dei giocatori con cui è connesso.
Protocolli e Attività	Consenso, Ordine.
Responsabilità	Deve compiere le sue scelte in modo ottimale.
Lifeness	Sempre.
Safety	Nessuno.

Tabella B.1: Ruolo Giocatore.

Ruolo	OppNatura
Descrizione	Modella come un giocatore con strategia aleatoria la domanda.
Permessi	Cambia la propria strategia.
Protocolli e Attività	Ordine.
Responsabilità	Deve compiere le scelte secondo un modello probabilistico.
Lifeness	Sempre.
Safety	Nessuno.

Tabella B.2: Ruolo OppNatura.

Protocollo	Consenso
Scopo	Essere informati delle scelte dei giocatori a cui si è connessi.
Initiator Responder	Giocatore. Giocatore.
ingressi uscite elaborazione	Nessuno. Il valore strategico al tempo $t-1$. Controlla se è connesso al giocatore.

Tabella B.3: Protocollo Consenso.

Protocollo	Ordine.
Scopo	Comunica la volontà di acquistare o meno.
Initiator Responder	OppNatura. Giocatore.
ingressi uscite elaborazione	Nessuno. Nessuno. Giocatore memorizza l'uscita che è associata alla sua azione e il payoff.

Tabella B.4: Protocollo Ordine.

Appendice C

Modello delle classi

In questo capitolo vengono documentate nei dettagli le classi usate nella simulazione unitamente agli attributi ed ai metodi. Il diagramma delle classi è rappresentato in fig.C.1.

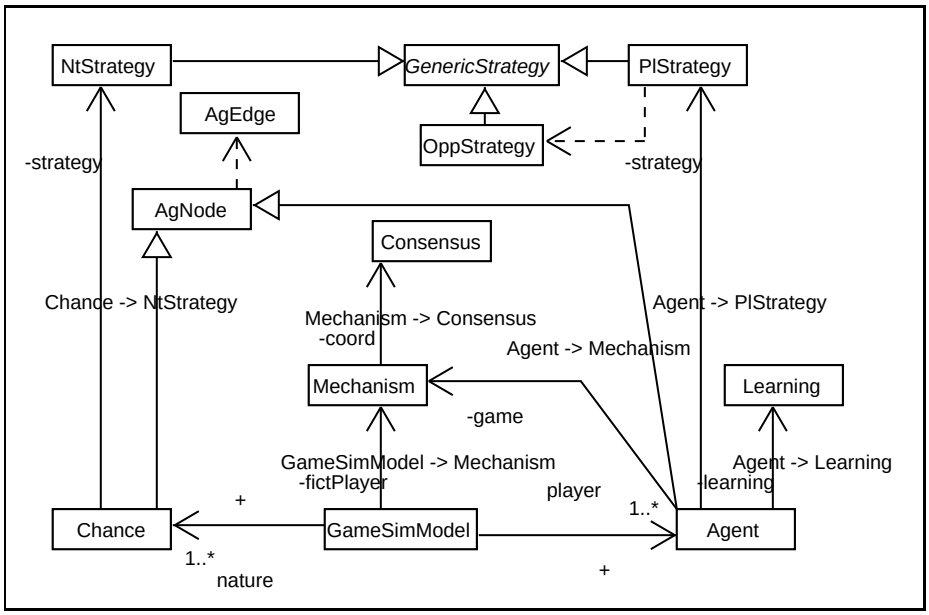

Figura C.1: Diagramma delle classi.

C.1 Riferimenti per la classe simtesi.GameSimModel

Membri pubblici

- **GameSimModel** ()
- String **getName** ()
- Schedule **getSchedule** ()
- String[] **getInitParam** ()
- double **getProbability** ()
- void **setProbability** (double newProb)
- int **getConnection** ()
- void **setConnection** (int newConn)
- boolean **getMemory** ()
- void **setMemory** (boolean mem)
- void **setH** (float newH)
- float **getH** ()
- void **setK** (float newK)
- float **getK** ()
- void **setEpsilon** (double newEpsilon)
- double **getEpsilon** ()
- void **setPlayers** (int n)
- int **getPlayers** ()
- void **setGamma** (double newGamma)
- double **getGamma** ()

- void **setup** ()
- void **begin** ()
- void **step** ()
- boolean **getMinimax** ()
- void **setMinimax** (boolean b)
- int **getBaseAmp** ()
- void **setBaseAmp** (int i)
- int **getAmplitudes** ()
- void **setAmplitudes** (int i)
- int **getAgent** ()
- void **setAgent** (int i)

Membri pubblici statici

- void **main** (String[] args)

Membri privati

- void **buildModel** ()
- void **buildGame** ()
- void **manageNtStrategies** ()
- void **buildDisplay** ()
- void **drawState** (final int n)
- void **drawAgState** (final int n)
- void **drawCost** (final int n)
- void **buildSchedule** ()

- boolean **errInput** ()
- void **manageConnection** ()
- void **createNetwork** ()
- void **disposeAgents** ()

Attributi privati

- double **prob** = 0.5
- Vector **player**
- Vector **nature**
- **Mechanism fictPlayer**
- int **numPlayers** = 8
- int **numNature** = **numPlayers**
- double **gamma** = 0.02
- float **h** = 3
- float **k** = 2
- double **epsilon** = 0.025
- boolean **memory** = false
- boolean **minimax** = true
- int **connection** = **Agent**.LOOP
- int **baseAmp** = -1
- int **amplitudes** = **Chance**.EQUAL
- AbstractGraphLayout **graphLayout**
- DisplaySurface **dsurf**
- Network2DGridDisplay **agDisplay**
- OpenSequenceGraph **graphCost**

- OpenSequenceGraph **graphState**
- OpenSequenceGraph **graphAgent**
- int **agent** = -1
- Schedule **schedule**
- int **fact** = -1

C.1.1 Descrizione Dettagliata

Classe principale per la simulazione della piattaforma.

Estende la classe SimModelImpl della piattaforma, e si occupa di istanziare i vari agenti e gestire la loro connessione a partire dalle opzioni scelte nell'interfaccia del gioco.

Ha la responsabilità di gestire: i paramentri della simulazione, la visualizzazione dei grafici e la ripetizione del gioco.

Autore:

Andrea Piran

C.1.2 Documentazione dei costruttori e dei distruttori

C.1.2.1 simtesi.GameSimModel.GameSimModel ()

Costruttore della classe.

C.1.3 Documentazione delle funzioni membro

C.1.3.1 void simtesi.GameSimModel.begin ()

Metodo per i settaggi da fare all'inizio della simulazione.

C.1.3.2 void simtesi.GameSimModel.buildDisplay () [private]

Costruisce i display per la visualizzazione dei grafici in uscita.

C.1.3.3 void simtesi.GameSimModel.buildGame () [private]

Metodo per la costruzione del gioco.

C.1.3.4 void simtesi.GameSimModel.buildModel () [private]

Metodo per la creazione degli oggetti creati nella simulazione.

C.1.3.5 void simtesi.GameSimModel.buildSchedule () [private]

Setta le azioni da compiere ad ogni partita del gioco.

C.1.3.6 void simtesi.GameSimModel.createNetwork () [private]

Crea la rete di agenti da visualizzare nel display.

C.1.3.7 void simtesi.GameSimModel.disposeAgents () [private]

Metodo per la disposizione degli agenti sulla superfice dello schermo.

C.1.3.8 void simtesi.GameSimModel.drawAgState (final int *n*) [private]

Metodo per la gestione della visualizzazione dello stato di un agente.

Parametri:

n Numero del giocatore di cui visualizzare lo stato.

C.1.3.9 void simtesi.GameSimModel.drawCost (final int *n*) [private]

Gestisce il grafico del costo medio.

Parametri:

n Numero degli agenti.

C.1.3.10 void simtesi.GameSimModel.drawState (final int *n*) [private]

Metodo per la gestione della visualizzazione dello stato medio.

Parametri:

n Numero dei giocatori.

C.1.3.11 boolean simtesi.GameSimModel.errInput () [private]

Controlla se i parametri dell'utente non contengono errori e/o incongruenze, se c'è un errore ritorna true e ferma la simulazione.

C.1.3.12 int simtesi.GameSimModel.getAgent ()

Restituisce l'agente del quale disegnare lo stato.

C.1.3.13 int simtesi.GameSimModel.getAmplitudes ()

Restituisce il tipo di ampiezza (i.e., multiple, uguali etc.) della strategia ad onda quadra per i giocatori natura.

C.1.3.14 int simtesi.GameSimModel.getBaseAmp ()

Restituisce l'ampiezza della strategia ad onda quadra del giocatore natura.

C.1.3.15 int simtesi.GameSimModel.getConnection ()

Ritorna il tipo di connessione.

C.1.3.16 double simtesi.GameSimModel.getEpsilon ()

Ritorna il fattore di sperimentazione.

C.1.3.17 double simtesi.GameSimModel.getGamma ()

Restituisce il valore del fattore d'apprendimento.

C.1.3.18 float simtesi.GameSimModel.getH ()

Ritorna il costo h.

C.1.3.19 String [] simtesi.GameSimModel.getInitParam ()

Restituisce i parametri della simulazione.

C.1.3.20 float simtesi.GameSimModel.getK ()

Ritorna il costo k.

C.1.3.21 boolean simtesi.GameSimModel.getMemory ()

Ritorna true se la pdf del giocatore natura è con memoria.

C.1.3.22 boolean simtesi.GameSimModel.getMinimax ()

Ritorna true se l'agente usa una strategia minimax.

C.1.3.23 String simtesi.GameSimModel.getName ()

Restituisce il nome della simulazione.

C.1.3.24 int simtesi.GameSimModel.getPlayers ()

Restituisce il numero dei giocatori.

C.1.3.25 double simtesi.GameSimModel.getProbability ()

Ritorna la probabilità del modello bernoulliano. la probabilità ritornata è quella relativa alla strategia "compro".

C.1.3.26 Schedule simtesi.GameSimModel.getSchedule ()

Restituisce l'oggetto con le azioni da compiere.

C.1.3.27 void simtesi.GameSimModel.main (String[] *args*) [static]

Metodo principale.

C.1.3.28 void simtesi.GameSimModel.manageConnection () [private]

Gestisce la connessione fra giocatori.

C.1.3.29 void simtesi.GameSimModel.manageNtStrategies () [private]

Metodo per la gestione delle ampiezze delle strategie "ad onda quadra" degli agenti di tipo **Chance**(pag. 159).

C.1.3.30 void simtesi.GameSimModel.setAgent (int *i*)

Setta l'agente di cui disegnare lo stato.

Parametri:

i Numero del giocatore.

C.1.3.31 void simtesi.GameSimModel.setAmplitudes (int *i*)

Setta il tipo di ampiezza della strategia ad onda quadra del giocatore natura.

Parametri:

i Tipo di ampiezza delle onde per gli altri giocatori.

C.1.3.32 void simtesi.GameSimModel.setBaseAmp (int *i*)

Setta l'ampiezza della strategia ad onda quadra del giocatore natura.

Parametri:

i Ampiezza dell'onda di base.

C.1.3.33 void simtesi.GameSimModel.setConnection (int *newConn*)

Setta il tipo di connessione.

Parametri:

newConn Nuova connessione dei giocatori.

C.1.3.34 void simtesi.GameSimModel.setEpsilon (double *newEpsilon*)

Setta il fattore di sperimentazione.

Parametri:

newEpsilon Nuovo valore di epsilon.

C.1.3.35 void simtesi.GameSimModel.setGamma (double *newGamma*)

Setta la velocità d'apprendimento. Un valore basso aumenta la precisione.

Parametri:

newGamma Nuovo valore di gamma.

C.1.3.36 void simtesi.GameSimModel.setH (float *newH*)

Setta il costo h.

Parametri:

newH Nuovo valore del costo h.

C.1.3.37 void simtesi.GameSimModel.setK (float *newK*)

Setta il costo k.

Parametri:

newK Nuovo valore del costo k.

C.1.3.38 void simtesi.GameSimModel.setMemory (boolean *mem*)

Setta la pdf del giocatore natura se con memoria o meno.

Parametri:

mem Se è true il modello è con memoria.

C.1.3.39 void simtesi.GameSimModel.setMinimax (boolean *b*)

Setta se l'agente usa la strategia minimax.

Parametri:

b Se è true la strategia utilizzata è quella minimax.

C.1.3.40 void simtesi.GameSimModel.setPlayers (int *n*)

Setta il numero dei giocatori.

Parametri:

n Numero dei giocatori.

C.1.3.41 void simtesi.GameSimModel.setProbability (double *newProb*)

Setta la probabilità del modello bernoulliano. E' la probabilità legata alla strategia "compro".

Parametri:

newProb Nuova probabilità del modello.

C.1.3.42 void simtesi.GameSimModel.setup ()

Metodo per il setup degli agenti.

C.1.3.43 void simtesi.GameSimModel.step ()

Metodo che definisce le azioni degli agenti ad ogni partita.

C.1.4 Documentazione dei dati membri

C.1.4.1 Network2DGridDisplay simtesi.GameSimModel.agDisplay [private]

Display su cui vengono disegnati gli agenti.

C.1.4.2 int simtesi.GameSimModel.agent = -1 [private]

Agente di cui disegnare lo stato.

C.1.4.3 int simtesi.GameSimModel.amplitudes = Chance.EQUAL [private]

Tipo di generazione delle ampiezze degli altri giocatori di tipo **Chance**(pag. 159).

C.1.4.4 int simtesi.GameSimModel.baseAmp = -1 [private]

Ampiezza base della strategia ad onda quadra del giocatore natura.

C.1.4.5 int simtesi.GameSimModel.connection = Agent.LOOP [private]

Tipo di connessione.

C.1.4.6 DisplaySurface simtesi.GameSimModel.dsurf [private]

Superfice del display.

C.1.4.7 double simtesi.GameSimModel.epsilon = 0.025 [private]

Fattore di sperimentazione.

C.1.4.8 int simtesi.GameSimModel.fact = -1 [private]

Fattore di scala usato per la generazione del display.

C.1.4.9 Mechanism simtesi.GameSimModel.fictPlayer [private]

Giocatore fittizio.

C.1.4.10 double simtesi.GameSimModel.gamma = 0.02 [private]

Fattore di apprendimento.

C.1.4.11 OpenSequenceGraph simtesi.GameSimModel.graphAgent [private]

Grafico di un agente.

C.1.4.12 OpenSequenceGraph simtesi.GameSimModel.graphCost [private]

Grafico del costo medio.

C.1.4.13 AbstractGraphLayout simtesi.GameSimModel.graphLayout [private]

Layout del display.

C.1.4.14 OpenSequenceGraph simtesi.GameSimModel.graphState [private]

Grafico dello stato medio.

C.1.4.15 float simtesi.GameSimModel.h = 3 [private]

Parametro h della matrice dei payoff.

C.1.4.16 float simtesi.GameSimModel.k = 2 [private]

Parametro k della matrice dei payoff.

C.1.4.17 boolean simtesi.GameSimModel.memory = false [private]

Vero se il metodo probabilistico dei giocatori natura è con memoria.

C.1.4.18 boolean simtesi.GameSimModel.minimax = true [private]

Se vero l'agente usa la strategia minimax.

C.1.4.19 Vector simtesi.GameSimModel.nature [private]

Lista dei giocatori che modellano la domanda.

C.1.4.20 int simtesi.GameSimModel.numNature = numPlayers [private]

Numero degli opponenti (Natura)

C.1.4.21 int simtesi.GameSimModel.numPlayers = 8 [private]

Numero degli agenti

C.1.4.22 Vector simtesi.GameSimModel.player [private]

Lista dei giocatori.

C.1.4.23 double simtesi.GameSimModel.prob = 0.5 [private]

Probabilità della strategia senza memoria dell'opponente natura.

C.1.4.24 Schedule simtesi.GameSimModel.schedule [private]

Attributo per schedulare le azioni degli agenti.

C.2 Riferimenti per la classe simtesi.Agent

Eredita da **simtesi.AgNode**.

Membri pubblici

- **Agent** (int i, float gamma, float epsilon)
- String **getOppStrategy** ()
- void **addStrategy** (**PlStrategy** action)
- Vector **getConnectedValue** ()
- int **getIndexStrategy** (String s)
- Vector **getLStrategy** ()
- Integer **getName** ()
- float **getPayoff** ()
- float **getPayoff** (int i)
- float **getProbability** (int i)
- **PlStrategy getLstStrategy** ()
- int **getStrategy** ()
- void **play** ()
- void **setConnectedValue** (Integer n)
- void **setPayoff** (float payoff)
- void **setProbability** (float prob, int i)
- void **setStrategy** (int newStrategy)
- void **setGame** (**Mechanism** newGame)
- float **getMediumPayoff** ()
- Vector **getLstCons** ()

- void **setLstCons** (Vector newPCons)
- int **getOldStrategy** ()
- boolean **isMinimax** ()
- void **setMinimax** (boolean b)
- int **getNPlay** ()
- boolean **isFixed** ()
- void **setFixed** (boolean b)
- void **setGamePayoff** ()

Attributi pubblici statici

- final int **CHAIN** = 0
- final int **LOOP** = 1
- final int **COMPLETE** = 2
- final int **STAR** = 3

Membri privati

- void **updateKnlGame** ()
- int **evaluateStrategy** ()
- int **getCons** ()

Attributi privati

- int **consValue**
- int **oldStrategy**
- Integer **name**

- Vector **lstStrategy**
- Vector **lstConn**
- Vector **lstCons**
- **PlStrategy strategy**
- **Learning learning**
- **Mechanism game**
- int **nPlay**
- float **totalPayoff**
- boolean **fixed** = false
- String **oppStrategy**
- boolean **minimax**

C.2.1 Descrizione Dettagliata

La classe modella l'agente del gioco.

L'agente è programmato implementando l'interfaccia GameAgent della piattaforma RePast e aggiorna le probabilità associate alla strategia mista secondo l'algoritmo RLA implementato nella classe **Learning**(pag. 178).

L'aggiornamento è necessario perché si ipotizza che l'agente inizi a giocare conoscendo solo le mosse che può fare ma non i pagamenti associati.

Autore:

Andrea Piran

C.2.2 Documentazione dei costruttori e dei distruttori

C.2.2.1 simtesi.Agent.Agent (int *i*, float *gamma*, float *epsilon*)

Costruttore della classe.

Parametri:

i Nome dell'agente.

gamma Valore del parametro gamma.

epsilon Valore del parametro epsilon.

C.2.3 Documentazione delle funzioni membro

C.2.3.1 void simtesi.Agent.addStrategy (PlStrategy *action*)

Aggiunge una strategia nella lista delle strategie del giocatore.

Parametri:

action Strategia da aggiungere alla lista.

C.2.3.2 int simtesi.Agent.evaluateStrategy () `[private]`

Metodo per la valutazione della migliore strategia.

C.2.3.3 Vector simtesi.Agent.getConnectedValue ()

Ritorna i valori dei giocatori connessi.

C.2.3.4 int simtesi.Agent.getCons () `[private]`

Ritorna il valore del protocollo di consenso.

C.2.3.5 int simtesi.Agent.getIndexStrategy (String *s*)

Restituisce l'indice della strategia utilizzata.

Parametri:

s Nome della strategia.

C.2.3.6 Vector simtesi.Agent.getLstCons ()

Restituisce la lista delle strategie dei giocatori a cui è connesso.

C.2.3.7 Vector simtesi.Agent.getLStrategy ()

Ritorna la lista delle strategie note all'Agente.

C.2.3.8 PlStrategy simtesi.Agent.getLstStrategy ()

Ritorna la strategia utilizzata.

C.2.3.9 float simtesi.Agent.getMediumPayoff ()

ritorna il payoff medio dell'agente.

C.2.3.10 Integer simtesi.Agent.getName ()

Ritorna il nome del giocatore.

C.2.3.11 int simtesi.Agent.getNPlay ()

Ritorna il numero di partite effettuate.

C.2.3.12 int simtesi.Agent.getOldStrategy ()

Ritorna la strategia al tempo t-1.

C.2.3.13 String simtesi.Agent.getOppStrategy ()

Ritorna la strategia giocata dall'avversario del giocatore.

C.2.3.14 float simtesi.Agent.getPayoff (int *i*)

Restituisce il valore associato all'azione i.

Parametri:

i Indice della strategia.

C.2.3.15 float simtesi.Agent.getPayoff ()

Restituisce il valore dell'utilità.

C.2.3.16 float simtesi.Agent.getProbability (int *i*)

Ritorna la probabilità associata alla strategia i.

Parametri:

i Indice della strategia.

C.2.3.17 int simtesi.Agent.getStrategy ()

Restituisce la strategia corrente.

C.2.3.18 boolean simtesi.Agent.isFixed ()

Se è true il giocatore è forzato allo stato "non produco".

C.2.3.19 boolean simtesi.Agent.isMinimax ()

Ritorna true se l'agente usa una strategia minimax.

C.2.3.20 void simtesi.Agent.play ()

Metodo che richiama le mosse del gioco.

C.2.3.21 void simtesi.Agent.setConnectedValue (Integer *n*)

Setta i valori della connessione fra giocatori.

Parametri:

n Nome dell'agente da aggiungere alla lista.

C.2.3.22 void simtesi.Agent.setFixed (boolean *b*)

Setta se il giocatore dev'essere forzato allo stato "non produco".

Parametri:

b Se è true, alla partita successiva lo stato dell'agente è "non produco".

C.2.3.23 void simtesi.Agent.setGame (Mechanism *newGame*)

Setta il giocatore fittizio.

Parametri:

newGame Giocatore fittizio del gioco.

C.2.3.24 void simtesi.Agent.setGamePayoff ()

Metodo per la gestione del payoff del gioco.

Viene usato anche per chiamare l'aggiornamento delle probabilità e la ricostruzione del gioco.

C.2.3.25 void simtesi.Agent.setLstCons (Vector *newPCons*)

Setta la liste delle strategie dei giocatori a cui è connesso.

Parametri:

newPCons Nuova lista delle strategie.

C.2.3.26 void simtesi.Agent.setMinimax (boolean *b*)

Setta se l'agente usa la strategia minimax.

Parametri:

b Se è true la strategia utilizzata è quella minimax.

C.2.3.27 void simtesi.Agent.setPayoff (float *payoff*)

Setta l'utiltà per l'agente.

Parametri:

payoff Payoff della strategia.

C.2.3.28 void simtesi.Agent.setProbability (float *prob*, int *i*)

Setta la probabilità associata alla strategia i.

Parametri:

prob Probabilità della strategia.

i Indice della strategia.

C.2.3.29 void simtesi.Agent.setStrategy (int *newStrategy*)

Setta la strategia per l'agente.

Parametri:

newStrategy Strategia dell'agente.

C.2.3.30 void simtesi.Agent.updateKnlGame () [private]

Aggiorna la conoscenza del gioco.

C.2.4 Documentazione dei dati membri

C.2.4.1 final int simtesi.Agent.CHAIN = 0 [static]

Connessione a catena.

C.2.4.2 final int simtesi.Agent.COMPLETE = 2 [static]

Connessione completa.

C.2.4.3 int simtesi.Agent.consValue [private]

Valore di consenso. Rappresenta lo stato medio degli agenti ed è una stima di n(t).

C.2.4.4 boolean simtesi.Agent.fixed = false [private]

Se è true al tempo t lo stato natura è "compro", quindi l'agente è forzato allo stato "non produco".

C.2.4.5 Mechanism simtesi.Agent.game [private]

Giocatore fittizio.

C.2.4.6 Learning simtesi.Agent.learning [private]

Apprendimento del giocatore.

C.2.4.7 final int simtesi.Agent.LOOP = 1 [static]

Connessione ad anello.

C.2.4.8 Vector simtesi.Agent.lstConn [private]

Lista dei giocatori con cui è connesso.

C.2.4.9 Vector simtesi.Agent.lstCons [private]

Lista delle strategie comunicate.

C.2.4.10 Vector simtesi.Agent.lstStrategy [private]

Lista delle strategie del giocatore.

C.2.4.11 boolean simtesi.Agent.minimax [private]

Se è true l'agente usa la strategia minimax.

C.2.4.12 Integer simtesi.Agent.name [private]

Nome del giocatore: è un numero progressivo. Name è un attributo di classe integer per motivi implementativi della classe Vector.

C.2.4.13 int simtesi.Agent.nPlay [private]

Contatore delle partite.

C.2.4.14 int simtesi.Agent.oldStrategy [private]

Strategia al tempo t-1.

C.2.4.15 String simtesi.Agent.oppStrategy [private]

Strategia dell'avversario.

C.2.4.16 final int simtesi.Agent.STAR = 3 [static]

Connessione a stella.

C.2.4.17 PlStrategy simtesi.Agent.strategy [private]

Strategia del giocatore.

C.2.4.18 float simtesi.Agent.totalPayoff [private]

payoff complessivo.

C.3 Riferimenti per la classe simtesi.Chance

Eredita da **simtesi.AgNode**.

Membri pubblici

- **Chance** (int i, boolean mem, double newProb)
- void **play** ()
- float **getPayoff** ()
- int **getIndexStrategy** (String s)
- int **getStrategy** ()
- void **setPayoff** (float payoff)
- void **setStrategy** (int newStrategy)
- void **addStrategy** (**NtStrategy** action)
- String **getNStrategy** ()
- int **getAmplitude** ()
- void **setAmplitude** (int i)

Attributi pubblici statici

- final int **EQUAL** = 0
- final int **SUBMULT** = 1
- final int **MULT** = 2
- final int **RANDOM** = 3

Membri privati

- void **evaluateStrategy** ()
- float **updateProbability** (double alpha)
- int **squareStrategy** (int amp)

Attributi privati

- Vector **nStrategy**
- **NtStrategy strategy**
- int **name**
- boolean **memory**
- double **gOld** = -Math.log(0.5)
- double **prob**
- int **nPlay**
- int **amplitude**

C.3.1 Descrizione Dettagliata

Giocatore che gioca con strategia aleatoria che modella la natura.

L'agente è programmato implementando l'interfaccia GameAgent della piattaforma RePast. I metodi getPayoff e setPayoff sono definiti per comformarsi a tale interfaccia, il giocatore gioca una strategia modellata con una distribuzione aleatoria che può essere con memoria oppure senza memoria.

Autore:

Andrea Piran

C.3.2 Documentazione dei costruttori e dei distruttori

C.3.2.1 simtesi.Chance.Chance (int *i*, boolean *mem*, double *newProb*)

Costruttore della classe.

Parametri:

i Nome dell'agente.

mem Se è true usa la strategia con memoria.

newProb Probabilità iniziale del modello probabilistico.

C.3.3 Documentazione delle funzioni membro

C.3.3.1 void simtesi.Chance.addStrategy (NtStrategy *action*)

Aggiunde una strategia alla lista delle strategie.

Parametri:

action Strategia da aggiungere.

C.3.3.2 void simtesi.Chance.evaluateStrategy () `[private]`

Metodo per la generazione della strategia.

C.3.3.3 int simtesi.Chance.getAmplitude ()

Ritorna l'ampiezza della strategia ad onda quadra.

C.3.3.4 int simtesi.Chance.getIndexStrategy (String *s*)

Restituisce l'indice della strategia utilizzata.

Parametri:

s Nome della strategia.

C.3.3.5 String simtesi.Chance.getNStrategy ()

Ritorna il nome della strategia giocata.

C.3.3.6 float simtesi.Chance.getPayoff ()

Restituisce il valore dell'utilità. Anche se non ha senso per l'agente in questione viene inserito poiché implementa un'interfaccia.

C.3.3.7 int simtesi.Chance.getStrategy ()

Restituisce la strategia corrente.

C.3.3.8 void simtesi.Chance.play ()

Metodo che richiama le mosse del gioco.

C.3.3.9 void simtesi.Chance.setAmplitude (int *i*)

Setta l'ampiezza dell'onda quadra.

C.3.3.10 void simtesi.Chance.setPayoff (float *payoff*)

Setta l'utiltà per l'agente. Anche se non ha senso per l'agente in questione viene inserito poiché implementa un'interfaccia.

C.3.3.11 void simtesi.Chance.setStrategy (int *newStrategy*)

Setta la strategia per l'agente.

C.3.3.12 int simtesi.Chance.squareStrategy (int *amp*) `[private]`

Genera la strategia ad onda quadra. Il parametro n è l'ampiezza del periodo. Avendo un contatore delle partite, se la divisione fra numero di partite ed ampiezza ha resto zero su ha la transizione.

Parametri:

amp Ampiezza della strategia.

C.3.3.13 float simtesi.Chance.updateProbability (double *alpha*) `[private]`

Aggiorna le probabilità nel caso di giocatore con memoria secondo l'equazione: p(x)=e^-g(t) ; g(t)=(1-alpha)∗g(t-1)+alpha∗x(t-1).

Parametri:

alpha valore del parametro alpha.

C.3.4 Documentazione dei dati membri

C.3.4.1 int simtesi.Chance.amplitude [private]

Ampiezza della strategia ad onda quadra.

C.3.4.2 final int simtesi.Chance.EQUAL = 0 [static]

Ampiezze uguali per tutte le strategie ad onda quadra.

C.3.4.3 double simtesi.Chance.gOld = -Math.log(0.5) [private]

Valore al passo precedente dell'esponente legato alla probabilità.

C.3.4.4 boolean simtesi.Chance.memory [private]

True se il modello probabilistico della strategia è di tipo con memoria.

C.3.4.5 final int simtesi.Chance.MULT = 2 [static]

Ampiezze multiple di quella base per tutte le strategie ad onda quadra.

C.3.4.6 int simtesi.Chance.name [private]

Numero del giocatore casuale.

C.3.4.7 int simtesi.Chance.nPlay [private]

Contatore delle partite.

C.3.4.8 Vector simtesi.Chance.nStrategy [private]

Lista delle strategie.

C.3.4.9 double simtesi.Chance.prob [private]

Probabilità della strategia

C.3.4.10 final int simtesi.Chance.RANDOM = 3 [static]

Ampiezze generate in modo casuale rispetto a quella base per tutte le strategie ad onda quadra.

C.3.4.11 NtStrategy simtesi.Chance.strategy [private]

Strategia del giocatore.

C.3.4.12 final int simtesi.Chance.SUBMULT = 1 [static]

Ampiezze sottomultiple di quella base per tutte le strategie ad onda quadra.

C.4 Riferimenti per la classe simtesi.GenericStrategy

Base per **simtesi.NtStrategy**, **simtesi.OppStrategy**, e **simtesi.PlStrategy**.

Membri pubblici

- String **getName** ()
- void **setName** (String newName)
- float **getProbability** ()
- void **setProbability** (float prob)

Attributi protetti

- String **name**
- float **probability**

C.4.1 Descrizione Dettagliata

Classe base per la strategia del giocatore.

Modella come classe astratta il concetto di strategia definito mediante nome e probabilità associata le sottoclassi **PlStrategy**(pag. 169), **NtStrategy**(pag. 173) ed **OppStrategy**(pag. 175) la estendono con gli attributi necessari per gli agenti.

Autore:

Andrea Piran

C.4.2 Documentazione delle funzioni membro

C.4.2.1 String simtesi.GenericStrategy.getName ()

Ritorna il nome della strategia giocata.

Reimplementata in **simtesi.OppStrategy** (pag. 176).

C.4.2.2 float simtesi.GenericStrategy.getProbability ()

Ritorna il nome della probabilita della strategia giocata.

C.4.2.3 void simtesi.GenericStrategy.setName (String *newName*)

setta il nome della strategia.

Parametri:

newName Nome della strategia.

Reimplementata in **simtesi.OppStrategy** (pag. 176).

C.4.2.4 void simtesi.GenericStrategy.setProbability (float *prob*)

Setta il nome della probabilita della strategia giocata.

Parametri:

prob Valore della probabilità.

C.4.3 Documentazione dei dati membri

C.4.3.1 String simtesi.GenericStrategy.name [protected]

Nome della strategia.

C.4.3.2 float simtesi.GenericStrategy.probability [protected]

Probabilità della strategia.

C.5 Riferimenti per la classe simtesi.PlStrategy

Eredita da **simtesi.GenericStrategy**.

Membri pubblici

- **PlStrategy** (float poff, String nam, float prob)
- void **addOppStrategy** (String **name**, float **payoff**)
- Vector **getLOppStrategy** ()
- **OppStrategy getOppStrategy** (int i)
- float **getPayoff** ()
- void **setPayoff** (float newPayoff)
- void **setStrategy** (float poff, String nam, float prob)

Attributi privati

- float **payoff**
- Vector **lstOppStr**

C.5.1 Descrizione Dettagliata

Strategia del giocatore.

Differisce dalla classe **NtStrategy**(pag. 173), anch'essa derivata dalla classe **GenericStrategy**(pag. 166), per il fatto che nella strategia di un giocatore deterministico, ai fini dell'aggiornamento delle probabilità associate alla

strategia, deve essere definito un attributo relativo al payoff associato alla strategia.

Autore:

Andrea Piran

C.5.2 Documentazione dei costruttori e dei distruttori

C.5.2.1 simtesi.PlStrategy.PlStrategy (float *poff*, String *nam*, float *prob*)

Costruttore della classse.

Parametri:

poff Payoff.

nam Nome della strategia.

prob Probabilità associata.

C.5.3 Documentazione delle funzioni membro

C.5.3.1 void simtesi.PlStrategy.addOppStrategy (String *name*, float *payoff*)

Aggiunge una strategia alla lista delle strategie dell'avversario.

Parametri:

name Nome della strategia.

payoff Payoff associato.

C.5.3.2 Vector simtesi.PlStrategy.getLOppStrategy ()

Restituisce la lista delle strategie.

C.5.3.3 OppStrategy simtesi.PlStrategy.getOppStrategy (int *i*)

Restituisce la strategia dell'avversario.

Parametri:

i Indice della strategia.

C.5.3.4 float simtesi.PlStrategy.getPayoff ()

Ritorna il payoff della strategia.

C.5.3.5 void simtesi.PlStrategy.setPayoff (float *newPayoff*)

Setta il payoff della strategia.

Parametri:

newPayoff Valore del payoff.

C.5.3.6 void simtesi.PlStrategy.setStrategy (float *poff*, String *nam*, float *prob*)

Setta la strategia.

Parametri:

poff Payoff della strategia.

nam Nome della strategia.

prob Probabilità associata.

C.5.4 Documentazione dei dati membri

C.5.4.1 Vector simtesi.PlStrategy.lstOppStr [private]

Lista delle strategie del giocatore natura.

C.5.4.2 float simtesi.PlStrategy.payoff [private]

Payoff della strategia.

C.6 Riferimenti per la classe simtesi.NtStrategy

Eredita da **simtesi.GenericStrategy**.

Membri pubblici

- **NtStrategy** (String nam, float prob)
- void **setStrategy** (String nam, float prob)

C.6.1 Descrizione Dettagliata

La classe estende la classe base **GenericStrategy**(pag. 166) che modella la strategia di un giocatore.

In quanto strategia di un giocatore che usa una strategia di tipo aleatorio non ha necessità di playoff perché non è usata una valutazione di quest'ultimo.

Autore:

Andrea Piran

C.6.2 Documentazione dei costruttori e dei distruttori

C.6.2.1 simtesi.NtStrategy.NtStrategy (String *nam*, float *prob*)

Costruttore della classe.

Parametri:

nam Nome della strategia.

prob Probabilità associata.

C.6.3 Documentazione delle funzioni membro

C.6.3.1 void simtesi.NtStrategy.setStrategy (String *nam*, float *prob*)

Setta la strategia.

Parametri:

nam Nome della strategia.

prob Probabilità associata.

C.7 Riferimenti per la classe simtesi.OppStrategy

Eredita da **simtesi.GenericStrategy**.

Membri pubblici

- **OppStrategy** (String newName, float newPayoff)
- void **setName** (String newName)
- void **setPayoff** (float newPayoff)
- String **getName** ()
- double **getPayoff** ()
- boolean **isVariabile** ()
- void **setVariabile** (boolean value)

Attributi privati

- float **payoff**
- boolean **variabile**

C.7.1 Descrizione Dettagliata

Strategia dell'avversario. Viene usato come nodo figlio nell'albero di ricostruzione del gioco.

Autore:

Andrea Piran

C.7.2 Documentazione dei costruttori e dei distruttori

C.7.2.1 simtesi.OppStrategy.OppStrategy (String *newName*, float *newPayoff*)

Costruttore della classe

Parametri:

newName Nome della strategia.

newPayoff Payoff associato.

C.7.3 Documentazione delle funzioni membro

C.7.3.1 String simtesi.OppStrategy.getName ()

Restituisce il nome della strategia.

Reimplementa **simtesi.GenericStrategy** (pag. 167).

C.7.3.2 double simtesi.OppStrategy.getPayoff ()

Restituisce il valore del payoff della strategia.

C.7.3.3 boolean simtesi.OppStrategy.isVariabile ()

Ritorna se il costo è variabile.

C.7.3.4 void simtesi.OppStrategy.setName (String *newName*)

Setta il nome della strategia.

Parametri:

newName Nome della strategia.

Reimplementa **simtesi.GenericStrategy** (pag. 167).

C.7.3.5 void simtesi.OppStrategy.setPayoff (float *newPayoff*)

Setta il payoff della strategia.

Parametri:

newPayoff Valore della strategia.

C.7.3.6 void simtesi.OppStrategy.setVariabile (boolean *value*)

Setta il costo come variabile o meno.

Parametri:

value Se è true il costo è variabile.

C.7.4 Documentazione dei dati membri

C.7.4.1 float simtesi.OppStrategy.payoff [private]

Payoff della strategia.

C.7.4.2 boolean simtesi.OppStrategy.variabile [private]

Se è true il costo è variabile.

C.8 Riferimenti per la classe simtesi.Learning

Membri pubblici

- **Learning** (float sGamma, float sEpsilon)
- float **getEpsilon** ()
- float **getGamma** ()
- void **rla** (**Agent** ag, float **gamma**, float **epsilon**)
- void **setGamma** (float newGamma)
- void **setEpsilon** (float newEpsilon)

Attributi privati

- float **epsilon**
- float **gamma**

C.8.1 Descrizione Dettagliata

Classe che incapsula i metodi per l'apprendimento delle strategie.

Il metodo principale della classe è RLA(...) che implementa l'algoritmo omonimo descritto in [Fried96] che viene utilizzato impostando la probabilita iniziale legata alle strategie a 0.5.

C.8.2 Documentazione dei costruttori e dei distruttori

C.8.2.1 simtesi.Learning.Learning (float *sGamma*, float *sEpsilon*)

Costruttore della classe.

Parametri:

sGamma Valore di gamma.

sEpsilon Valore di epsilon.

C.8.3 Documentazione delle funzioni membro

C.8.3.1 float simtesi.Learning.getEpsilon ()

Ritorna il fattore di sperimentazione.

C.8.3.2 float simtesi.Learning.getGamma ()

Restituisce il fattore di apprendimento.

C.8.3.3 void simtesi.Learning.rla (Agent *ag*, float *gamma*, float *epsilon*)

Metodo per l'aggiornamento delle probabilità associate alle strategie. Implementa l'algoritmo RLA descritto in [Fried96].

Parametri:

ag

gamma

epsilon

C.8.3.4 void simtesi.Learning.setEpsilon (float *newEpsilon*)

Setta il fattore di apprendimento.

Parametri:

newEpsilon Valore di epsilon.

C.8.3.5 void simtesi.Learning.setGamma (float *newGamma*)

Modifica il fattore di apprendimento.

Parametri:

newGamma Valore di gamma.

C.8.4 Documentazione dei dati membri

C.8.4.1 float simtesi.Learning.epsilon [private]

Fattore di sperimentazione.

C.8.4.2 float simtesi.Learning.gamma [private]

Fattore di aggiornamento

C.9 Riferimenti per la classe simtesi.Mechanism

Membri pubblici

- **Mechanism** (Vector team, Vector opponents, double kNew)
- void **setPayoff** (float arg0)
- float **getPayoff** ()
- float **getPayoff** (Integer name)
- int **getStrategy** ()
- void **setStrategy** (int arg0)
- void **play** ()
- Vector **createPayoff** (double h, double **k**)
- void **modifyPayoff** ()
- void **createC** (Vector players)
- double **getSecEigen** ()
- void **getL** ()
- void **dualProblem** (**Agent** ag, boolean direct)
- Vector **getCsStrategy** (**Agent** ag, Vector connected)
- String **getOppStrategy** (**Agent** ag)

Membri privati

- Float **commPayoff** (**Agent** ag, **Chance** ng)

Attributi privati

- Vector **agents**
- Vector **chances**
- Vector **payoff**
- double **k**
- **Consensus coord**

C.9.1 Descrizione Dettagliata

Giocatore fittizio che implementa il gioco.

Viene modellato come un agente poiché, per ipotesi, l'ambiente ha una struttura che varia col tempo. L'agente ha la responsabilità di gestire le comunicazioni dei payoff e del consenso tra i giocatori.

L'agente non nasce dall'analisi dei requisiti poiché in quella sede non era evidente la sua necessità, ma nasce per motivi di semplificazione nella gestione delle modifiche strategiche legate ai risultati del gioco.

Autore:

Andrea Piran

C.9.2 Documentazione dei costruttori e dei distruttori

C.9.2.1 simtesi.Mechanism.Mechanism (Vector *team*, Vector *opponents*, double *kNew*)

Costruttore della classe.

Parametri:

team Lista dei giocatori.

opponents Lista degli avversari.

kNew Valore del costo k.

C.9.3 Documentazione delle funzioni membro

C.9.3.1 Float simtesi.Mechanism.commPayoff (Agent *ag*, Chance *ng*) [private]

Comunica i payoff delle strategie.

Parametri:

ag Giocatore.

ng Avversario.

C.9.3.2 void simtesi.Mechanism.createC (Vector *players*)

Aggiorna la matrice C.

Parametri:

players Lista dei giocatori.

C.9.3.3 Vector simtesi.Mechanism.createPayoff (double *h*, double *k*)

Setta i payoff del gioco.

Parametri:

h Costo h.

k Costo k.

C.9.3.4 void simtesi.Mechanism.dualProblem (Agent *ag*, boolean *direct*)

Crea il duale al problema di massimo. Se direct è al valore true, crea la rappresentazione duale, altrimenti ritorna il problema primario.

Parametri:

ag Giocatore.

direct Se è true fa la trasformazione diretta.

C.9.3.5 Vector simtesi.Mechanism.getCsStrategy (Agent *ag*, Vector *connected*)

Comunica al giocatore la strategia dei giocatori connessi.

Parametri:

ag Giocatore.

connected Lista dei giocatori cui è connesso il giocatore.

C.9.3.6 void simtesi.Mechanism.getL ()

Crea la matrice L

C.9.3.7 String simtesi.Mechanism.getOppStrategy (Agent *ag*)

Ritorna la strategia giocata dall'avversario di tipo natura.

Parametri:

ag Giocatore.

C.9.3.8 float simtesi.Mechanism.getPayoff (Integer *name*)

Ritorna il payoff all'agente che lo richiede.

Parametri:

name Nome dell'agente.

C.9.3.9 float simtesi.Mechanism.getPayoff ()

Metodo definito per conformarsi all'interfaccia. Non è usato perché è un giocatore fittizio.

C.9.3.10 double simtesi.Mechanism.getSecEigen ()

Ritorna il secondo autovalore più piccolo del laplaciano.

C.9.3.11 int simtesi.Mechanism.getStrategy ()

Metodo definito per conformarsi all'interfaccia. Non è usato perché è un giocatore fittizio.

C.9.3.12 void simtesi.Mechanism.modifyPayoff ()

Modifica la lista dei payoff.

Viene chiamato ad ogni ripetizione del gioco perché deve calcolatre il rapporto k/n(t).

C.9.3.13 void simtesi.Mechanism.play ()

Azioni per il gioco.

C.9.3.14 void simtesi.Mechanism.setPayoff (float *arg0*)

Metodo definito per conformarsi all'interfaccia. Non è usato perché è un giocatore fittizio.

C.9.3.15 void simtesi.Mechanism.setStrategy (int *arg0*)

Metodo definito per conformarsi all'interfaccia. Non è usato perché è un giocatore fittizio.

C.9.4 Documentazione dei dati membri

C.9.4.1 Vector simtesi.Mechanism.agents [private]

Lista dei giocatori.

C.9.4.2 Vector simtesi.Mechanism.chances [private]

Lista dei giocatori natura.

C.9.4.3 Consensus simtesi.Mechanism.coord [private]

Oggetto di tipo consenso.

C.9.4.4 double simtesi.Mechanism.k [private]

Fattore k della matrice dei payoff.

C.9.4.5 Vector simtesi.Mechanism.payoff [private]

Payoff del gioco.

C.10 Riferimenti per la classe simtesi.Consensus

Membri pubblici

- **Consensus** (int P)
- void **createC** (int i, Vector agConn)
- DoubleMatrix1D **eigenL** ()
- double **getValue** ()
- void **setValue** (double sValue)
- DoubleMatrix2D **compL** ()
- void **setX** (int i, int **value**)
- int **getX** (int i)
- DoubleMatrix1D **xNew** (DoubleMatrix1D x)

Attributi privati

- double **value**
- DoubleMatrix2D **C**
- DoubleMatrix1D **X**
- int **dim**

C.10.1 Descrizione Dettagliata

Classe che modella il consenso tra gli agenti.

Autore:

Andrea Piran

C.10.2 Documentazione dei costruttori e dei distruttori

C.10.2.1 simtesi.Consensus.Consensus (int *P*)

Costruttore della classe.

Parametri:

P Numero dei giocatori.

C.10.3 Documentazione delle funzioni membro

C.10.3.1 DoubleMatrix2D simtesi.Consensus.compL ()

C.10.3.2 void simtesi.Consensus.createC (int *i*, Vector *agConn*)

Genera la matrice di consenso.

Parametri:

i Nome del giocatore.

agConn Lista degli agenti a cui è connesso il giocatore.

C.10.3.3 DoubleMatrix1D simtesi.Consensus.eigenL ()

Ritorna gli autovalori del laplaciano associato alla rete.

C.10.3.4 double simtesi.Consensus.getValue ()

Ritorna il valore del consenso.

C.10.3.5 int simtesi.Consensus.getX (int *i*)

Ritorna il valore associato all'agente X.

Parametri:

i Indice nel vettore.

C.10.3.6 void simtesi.Consensus.setValue (double *sValue*)

Setta il valore di consenso.

Parametri:

sValue Valore di consenso.

C.10.3.7 void simtesi.Consensus.setX (int *i*, int *value*)

Setta il valore del vettore X delle decisioni.

Parametri:

i Indice nel vettore.

value Valore della decisione.

C.10.3.8 DoubleMatrix1D simtesi.Consensus.xNew (DoubleMatrix1D *x*)

Calcola il valore di X all'istante t+1.

Parametri:

x Vettore x.

C.10.4 Documentazione dei dati membri

C.10.4.1 DoubleMatrix2D simtesi.Consensus.C `[private]`

Matrice C.

C.10.4.2 int simtesi.Consensus.dim [private]

Dimensione della matrice.

C.10.4.3 double simtesi.Consensus.value [private]

Valore del consenso.

C.10.4.4 DoubleMatrix1D simtesi.Consensus.X [private]

Vettore X delle decisioni.

C.11 Riferimenti per la classe simtesi.AgEdge

Membri pubblici

- **AgEdge** (Node from, Node to, Color c)
- void **draw** (SimGraphics g, int fromX, int toX, int fromY, int toY)

Attributi privati

- Color **color**

C.11.1 Descrizione Dettagliata

Una connessione fra due agenti rappresentato come un arco fra due nodi.

Autore:

Andrea Piran

C.11.2 Documentazione dei costruttori e dei distruttori

C.11.2.1 simtesi.AgEdge.AgEdge (Node *from*, Node *to*, Color *c*)

Costruttore della classe

Parametri:

from Nodo sorgente.

to Nodo destinazione.

c Colore.

C.11.3 Documentazione delle funzioni membro

C.11.3.1 void simtesi.AgEdge.draw (SimGraphics *g*, int *fromX*, int *toX*, int *fromY*, int *toY*)

Disegna l'arco fra due nodi.

Parametri:

g Oggetto grafico.

fromX Coordinata x di partenza.

toX Coordinata x d'arrivo.

fromY Coordinata y di partenza.

toY Coordinata y d'arrivo.

C.11.4 Documentazione dei dati membri

C.11.4.1 Color simtesi.AgEdge.color `[private]`

Colore dell'arco.

C.12 Riferimenti per la classe simtesi.AgNode

Base per **simtesi.Agent**, e **simtesi.Chance**.

Membri pubblici

- **AgNode** ()
- **AgNode** (int x, int y)
- void **init** (int x, int y)
- void **setPosition** (int x, int y)
- void **setDimension** (int dim)
- void **makeEdgeFromTo** (DefaultNode nodeIn, DefaultNode nodeOut, Color color)

Attributi con visibilità di package

- OvalNetworkItem **rect**

C.12.1 Descrizione Dettagliata

Un link fra due agenti.

Autore:

Andrea Piran

C.12.2 Documentazione dei costruttori e dei distruttori

C.12.2.1 simtesi.AgNode.AgNode ()

Costruttore della classe.

C.12.2.2 simtesi.AgNode.AgNode (int *x*, int *y*)

Costruttore della classe.

Parametri:

x Coordinata x.

y Coordinata y.

C.12.3 Documentazione delle funzioni membro

C.12.3.1 void simtesi.AgNode.init (int *x*, int *y*)

Inizializza il Nodo.

Parametri:

x Coordinata x.

y Coordinata y.

C.12.3.2 void simtesi.AgNode.makeEdgeFromTo (DefaultNode *nodeIn*, DefaultNode *nodeOut*, Color *color*)

Crea un arco fra due nodi.

Parametri:

nodeIn Nodo di partenza dell'arco.

nodeOut Nodo d'arrivo dell'arco.

color Colore dell'arco.

C.12.3.3 void simtesi.AgNode.setDimension (int *dim*)

Setta la dimensione del nodo.

Parametri:

dim Dimensione del nodo.

C.12.3.4 void simtesi.AgNode.setPosition (int *x*, int *y*)

Setta la posizione del nodo.

Parametri:

x Coordinata x.

y Coordinata y.

C.12.4 Documentazione dei dati membri

C.12.4.1 OvalNetworkItem simtesi.AgNode.rect [package]

Oggetto che viene mostrato nel display.

Bibliografia

[Fried96] E. J. Friedman, S. Shenker *Sincronous and Asynchronous Learning by Responsive Learning Automata*, working paper, ('96).

[Fried98] E. J. Friedman, S. Shenker *Learning and Implementation on the Internet*, working paper, ('98).

[Grenn00] A. Greenwald, E. J. Friedman, S. Shenker *Learning in Network Contexts: Experimental Results from Simulations*, working paper, ('00).

[Guat99] S. Guattery, G. L. Miller *Graph Embeddings and Laplacian Eigenvalues*, SIAM Journal of Matrix Analytical Applications, Vol.21 No.3 pp. 703-723, ('99).

[Guer03] F. Guerra, S. Arévalo, A. Alvarez, J. Miranda *A Distributed Consensus Protocol with a Coordinator*, technical paper, ('03).

[Hill94] F. S. Hiller, G. J. Liebermann *Introduzione alla Ricerca Operativa*, Franco Angeli, ('94).

[JV03] *Java 1.4 Documentation*, documentazione allegata al software, http://java.sun.com, ('03).

[Lamp82] L. Lamport, R. Shostak, M. Peace *The Byzantine Generls Problem*, ACM transactios on Programming Languages and Systems, vol.4 No.3 July 1982, Pages 382-401.

[Mohar91] B. Mohar *The Laplacian Spectrum of Graphs*, Graph Theory, Combinatorics, and Applications, Vol. 2, Ed. Y. Alavi, G. Chartrand, O. R. Oellerman, A. J. Schwenk, Wiley, pp. 871-898, ('91).

[Nilss01] N. J. Nilsson *Intelligenza Artificiale*, Apogeo, ('01).

[Olfa03] R. Olfati-Saber, R. M. Murray *Consensus Protocols for Networks of Dynamic Agents*, submitted to the American Control Conference, July 2003.

[Res90] M. D. Resnick *Scelte*, Muzzio Editore, ('90).

[RP03] *RePast 2.0.1 Documentation*, documentazione software, http://repast.sourceforge.net, ('03).

[Wool00] M. Woolridge, N. R. Jennings, D. Kinny *The Gaia Methodology for Agent-Oriented Analysis and Design*, Autonomous Agents and Multi-Agents Systems, Kluwer Academic Publishers, ('00).