INDICE

Introduzione

PARTE PRIMA

Economia tradizionale e economia sperimentale

1.1 Le radici storiche dell'economia sperimentale 6
1.2 Esperimenti e simulazioni 9
1.3 Alcune tipologie di esperimenti, le caratteristiche principali 11
1.4 L'uso delle simulazioni: vantaggi e svantaggi 14

PARTE SECONDA

Altruismo, egoismo, cooperazione e apprendimento

2.1 Altruismo ed egoismo 20
2.2 Homo Oeconomicus ed altruismo 21
2.3 Altruismo e razionalità 23
2.4 Altruismo ed evoluzione biologica 24
2.5 Altruismo, evoluzione culturale e norme sociali 26
2.6 Il tit for tat 27
2.7 Incertezza, razionalità limitata ed altruismo 28
2.8 Il ruolo dell'apprendimento 30
2.9 Altruismo ed emozioni 30
2.10 Il ruolo dell'altruismo nell'ambito economico 32

PARTE TERZA

Uno strumento per le simulazioni: Swarm

3.1 Il progetto Swarm 35
3.1.1 Una piattaforma dinamica 36

3.2 Swarm e la programmazione ad oggetti 37
3.2.1 Introduzione alla programmazione ad oggetti 38
3.3 Le librerie Swarm 40
3.3.1 Defobj Library 41
3.3.2 Collection Library 41
3.3.3 Activity Library 41
3.3.4 Objectbase Library 42
3.3.5 Random Library 42
3.3.6 Simtools Library 42
3.3.7 Gui Library 43
3.3.8 Analysis Library 43
3.3.9 Space Library 43
3.3.10 Probe Library 44
3.4 La struttura di una simulazione 44
3.4.1 Model Swarm 45
3.4.2 Observer Swarm 45
3.4.3 Main.m 46
3.4.4 RuleMaster 46
3.4.5 Batch Swarm 46
3.5 Alcuni oggetti importanti: liste, mappe e vettori 47
3.5.1 Le liste 47
3.5.2 I vettori 48
3.5.3 Le mappe 49

3.6 Schedules e Actions 50
3.6.1 Schedules 50
3.6.2 La classe Activity 51
3.7 L'interfaccia grafica di Swarm 51
3.8 Alcuni messaggi d'errore 55

PARTE QUARTA

Swarm Web Interface for Experimental Economics

4.1 Swiee: un'introduzione 58
4.2 Alcune caratteristiche di Swiee 59
4.2.1 La struttura di Swiee 60
4.3 Chiamate a metodi remoti (RMI) 62
4.4 Manuale di installazione di Swiee 64
4.4.1 Apache 65

4.4.2 Swarm 67
4.4.3 Java 2 Sdk 68
4.4.4 Autoexec.bat 68
4.5 Sequenza dei comandi per l'avvio dell'esperimento 68
4.6 Swiee e gli esperimenti sulla rete 69

PARTE QUINTA

Uno sguardo alla letteratura: i Classroom Game

5 Classroom game 72
5.1 Una "cascata" di informazioni 72
5.2 Approvvigionamento volontario di un bene pubblico 75
5.3 Comprendere la regola di Bayes 78
5.3.1 Un calcolo euristico 79
5.4 L'interazione strategica sull'internet 82
5.5 Un "mercato dei limoni" 84
5.6 Speculazioni e bolle sul mercato dei capitali 87
5.7 Generare moneta 90

PARTE SESTA

Gli esperimenti svolti con l'aiuto delle simulazioni

6.1 Gli esperimenti di Alessandria 96
6.1.1 L'analisi generale dei dati 103
6.1.2 Le diverse strategie 104
6.1.3 L'introduzione degli agenti artificiali 106
6.2 Il primo esperimento di Torino 107
6.2.1 Gli agenti artificiali 109
6.2.2 Modifiche al codice 111
6.2.3 Le strategie individuate 113
6.2.4 I preparativi 114
6.2.5 Alcuni problemi riscontrati 119
6.2.6 I dati raccolti 121
6.2.7 Analisi dei dati 130

6.3 Il secondo esperimento di Torino 139
6.3.1 Modifiche al codice 146
6.3.2 Altri problemi incontrati 148
6.4 Analisi dei risultati 152

CONCLUSIONE

APPENDICE

Listato dei programmi

INTRODUZIONE

L'economia sperimentale è una metodologia di analisi sviluppata recentemente e costituisce il tentativo da parte degli studiosi di colmare il divario esistente tra il mondo della teoria e della realtà. Tradizionalmente l'economia, disciplina che basa i suoi principi sull'osservazione di determinati fenomeni e sull'applicazione di teorie e modelli che descrivano tali fenomeni, ha fondato le basi sull'analisi di dati statistici derivanti dall'osservazione di mercati naturali e quindi situazioni reali. La complessità dei modelli teorici è, però, andata via via crescendo e la presenza di vincoli e condizioni sempre più precise e stringenti hanno reso la verifica delle teorie mediante dati storici sempre più difficile e limitata. E' nata così l'esigenza di studiare i modelli teorici attraverso osservazioni sperimentali. Uno degli aspetti più stimolanti dell'economia sperimentale è proprio la sua peculiare caratteristica di coniugare la teoria economica con l'econometria, l'analisi dei dati, l'ideazione di programmi, la programmazione e il contatto con la gente.

Nella prima parte, dopo una breve rassegna sulle radici storiche dell'economia sperimentale, si analizzeranno le analogie e le differenze di esperimenti e simulazioni; con l'introduzione di queste ultime, è stato possibile delegare, almeno in parte, ai computer il compito di derivare le predizioni empiriche provenienti dalla teoria. Lo scienziato, grazie al metodo della simulazione, è in grado di verificare in tempo reale se le sue teorie e le sue predizioni sono in linea con i risultati della sperimentazione.

Nel prosieguo del capitolo verranno presentate le principali tipologie di esperimenti: gioco con un singolo giocatore, a gruppi, sincrono e asincrono. Nell'ultima sezione si concentrerà l'attenzione sui vantaggi e sugli svantaggi dell'uso delle simulazioni come metodo di analisi. Emergeranno i benefici fondamentali di tale metodo che consente la replicabilità e il controllo delle variabili oggetto di studio e il limite relativo all'eccessiva semplificazione che tale metodo richiede.

Nella seconda parte, si tratterà il tema centrale del lavoro: l'esistenza o meno in economia di comportamenti altruistici e cooperativi. Il comportamento egoista, generalmente ipotizzato nell'ambito dei tradizionali modelli economici, pare smentito da un'ampia serie di osservazioni empiriche (Henrich ed altri, 2001); sarà quindi necessario ripercorrere le diverse idee proposte dalla letteratura per spiegare il sorgere di tali comportamenti. Si analizzerà l'ipotesi che l'altruismo derivi da processi di

selezione biologica o culturale e le relazioni con concetti quali emozioni, incertezza, razionalità e processi di apprendimento.

Nella terza parte della tesi analizzeremo gli strumenti occorsi per la creazione del programma utilizzato per i due esperimenti svolti a Torino: Swarm, un insieme di programmi e librerie standard, da impiegare per effettuare simulazioni e studiare sistemi complessi di comportamento, nell'ambito delle scienze naturali e sociali. Swarm permette di ricreare modelli di comportamento di un insieme di "agenti" (ABM - *Agent Based Modelling*) indipendenti, al fine di derivare - per mezzo di semplici regole - andamenti collettivi complessi. Tratteggeremo le caratteristiche principali della programmazione ad oggetti ed illustreremo le sue caratteristiche di astrazione, incapsulamento, ereditarietà e polimorfismo.

Nella quarta parte sarà studiato Swiee, un programma che permette di utilizzare Swarm per effettuare esperimenti in cui possono interagire esseri umani, o umani e agenti artificiali. Swiee è stato progettato adoperando delle classi RMI (chiamate a metodi remoti) di Java, classi che permettono ad agenti reali di interagire con il mondo artificiale e ai ricercatori di studiarne le conseguenze. Ogni agente umano prende parte all'esperimento effettuando le proprie scelte attraverso l'uso di un computer (*client*) collegato con il *server* che gestisce il programma. Il meccanismo RMI permette in particolare di richiamare i metodi di un oggetto che si trova su di un altro computer, in questo modo il *client* utilizza un *applet* presente sul *server* per svolgere l'esperimento.

La quinta parte propone un'analisi della, quantitativamente limitata, letteratura in ambito di esperimenti. Negli Stati Uniti il numero di docenti che utilizzano giochi di classe (*classroom game*) per spiegare agli studenti concetti e teorie fondamentali dell'economia sta crescendo. Dalla primavera del 1996 all'autunno del 2000 il "Journal of Economic Perspectives" ha presentato sette giochi che è possibile svolgere all'interno delle classi. Gli argomenti tema di tali esperienze sono: le informazioni a cascata, l'approvvigionamento volontario di un bene pubblico, comprendere la regola di Bayes, l'interazione strategica sull'internet, il "mercato dei limoni" (o delle truffe)[1], speculazioni e bolle sul mercato dei capitali e la generazione di moneta. Per ognuno di

[1] La definizione "mercato dei limoni" deriva da Gorge Akelrof (1970), il quale ha spiegato come la pressione determinata dalla competizione possa causare il deterioramento della qualità dei prodotti commerciati a livelli talmente bassi che il mercato potrebbe addirittura fallire (l'esempio più classico è quello del mercato delle autovetture usate).

questi temi si analizzerà brevemente la teoria sottostante e si spiegheranno le regole necessarie per lo svolgimento di un esperimento che consentirà di stimolare discussioni e approfondimenti oltre a permettere una più approfondita comprensione dei temi trattati.

L'ultima sezione è dedicata agli esperimenti: quello effettuato ad Alessandria con i soli agenti umani e quelli svolti a Torino, sede in cui si è deciso di inserire una fase di apprendimento con agenti artificiali per studiare eventuali influenze che tale apprendimento avrebbe potuto generare nel comportamento degli esseri umani.

L'esperimento di Alessandria si proponeva di analizzare il modo di agire degli esseri umani in situazioni predeterminate con un semplice gioco a somma 10. Il gioco prevedeva la formazione di squadre composte da tre giocatori umani che, investendo un valore scelto tra un set di numeri a disposizione, dovevano operare in modo tale da raggiungere la somma di 10 per ottenere il punteggio maggiore. Sono stati effettuati quattro turni con regole diverse per confrontare eventuali differenze nel comportamento, nate dalla consapevolezza o meno del numero dei turni a disposizione e nella presenza di un set di numeri fisso o variabile.

A Torino abbiamo approfondito l'argomento dell'altruismo e della cooperazione svolgendo due esperimenti simili a quelli effettuati ad Alessandria. Abbiamo introdotto due distinte fasi di gioco: la prima, con giocatori umani in squadra con agenti artificiali guidati da algoritmi, è servita come fase di apprendimento per cercare di "educare" i partecipanti a precisi comportamenti; la seconda, in cui le squadre erano composte da soli giocatori umani, ha fornito un ottimo banco di prova per verificare eventuali influenze nelle strategie scelte. Entrambi gli esperimenti fatti a Torino hanno fornito una serie interessante di dati che sono stati interpretati e analizzati con l'aiuto di grafici e tabelle.

In appendice sono riportati i listati dei programmi scritti in Java che abbiamo utilizzato per lo svolgimento dei due esperimenti svolti a Torino.

PARTE PRIMA

Economia tradizionale e economia sperimentale

1.1 Le radici storiche dell'economia sperimentale

Nonostante la formulazione di modelli teorici applicati alla struttura dei mercati risalga ai primi anni dell'800, l'economia sperimentale trova le sue origini nel 1931 con l'esperimento di Thurstone. Si tratta del primo esperimento di cui si sia a conoscenza, infatti come tale viene indicato nei lavori di ricostruzione della storia dell'economia sperimentale. Obiettivo di tale esperimento era di verificare la teoria neoclassica del consumatore[2] attraverso la determinazione empirica di una curva di indifferenza individuale.

Ad ogni soggetto coinvolto, Thurstone chiese di effettuare una ipotetica scelta tra panieri composti alternativamente di cappelli e scarpe oppure di cappelli e giacche oppure di scarpe e giacche. Trattandosi di scelte ipotetiche i soggetti non erano costretti ad acquisire effettivamente le combinazioni preferite. L'autore riportò i dati raccolti su un soggetto e stimò una curva di indifferenza, assunta di forma iperbolica, che si adattava alle osservazioni empiriche. La conclusione del lavoro di Thurnstone fu che le curve di indifferenza potevano essere stimate ed erano rispondenti ai risultati che emergevano dalle osservazioni empiriche.

Tra la fine degli anni '40 e l'inizio degli anni '50, si collocano invece i lavori di Chamberlin (1948), ripresi successivamente da V. Smith, sulle proprietà dell'equilibrio concorrenziale.

Nel 1953 Allais continua l'esplorazione con studi sul comportamento individuale in situazioni di rischio. Solo però con i lavori di Vernon Smith del 1962 ("An Experimental Study of Competitive Market Behaviour") l'economia sperimentale diventa una vera e propria metodologia di ricerca e negli anni '80 e '90 l'esperimento

[2] Nella teoria economica neoclassica il consumo rappresenta il mezzo per la soddisfazione dei bisogni, che risultano radicati nella natura umana. Il consumatore possiede un insieme di preferenze chiare e consapevoli, acquisite in via del tutto autonoma, senza subire l'influenza o l'autorità di altri soggetti. Sulla base di queste preferenze e delle risorse di cui dispone, l'individuo massimizza la soddisfazione dei propri bisogni (utilità) attraverso l'acquisto di una determinata combinazione di beni (domanda individuale). In questo quadro, la sovranità del consumatore non può che essere completa.

viene finalmente riconosciuto come un nuovo potente strumento per la ricerca applicata.

I filoni teorici di applicazione della metodologia sperimentale sono fondamentalmente tre. Il primo, che ha preso spunto dagli studi di Chamberlin, presta attenzione alla teoria neoclassica dei prezzi[3] e, in generale, al funzionamento dei mercati perfettamente concorrenziali. Il secondo, studiato e in parte sviluppato in questa tesi, analizza le implicazioni in termini di comportamento della teoria dei giochi e della contrattazione. Infine oggetto del terzo filone di studi è l'analisi del contenuto degli assiomi portanti della teoria dell'utilità attesa.[4]

Chamberlin sviluppò il suo primo esperimento di mercato nel 1948. Distribuì ai suoi giocatori, degli studenti, un mazzo di carte contrassegnate con valori e costi. I venditori, scambiando, potevano guadagnare la differenza tra il valore indicato dalla carta e il prezzo del contratto da loro negoziato, analogamente i compratori. I guadagni di tale esperimento erano del tutto ipotetici, ma i giocatori risultavano in qualche modo incentivati ed il processo dava origine ad una specifica forma di mercato. I risultati ottenuti si discostavano però sistematicamente dalle predizioni della teoria competitiva.

Solo alcuni anni dopo, Vernon Smith, che aveva partecipato all'esperimento di Chamberlin come studente, riprese in mano e riformulò l'intero esperimento convinto che le distorsioni rilevate dipendessero esclusivamente dal contesto[5] ipotizzato da Chamberlin stesso. Smith rielaborò pertanto l'esperimento introducendo il meccanismo dell'asta doppia nella quale tutte le offerte, sia quelle potenziali che quelle effettive, le domande (potenziali ed effettive) e le transazioni avvenute venivano rese note

---

[3] Obiettivo della teoria neoclassica è illustrare come l'utilità del sistema dei prezzi sia di consentire l'interdipendenza dei soggetti economici, dove per interdipendenza si intenda produzione volta a soddisfare la domanda di consumo altrui.

[4] La teoria normativa della decisione ha il suo fulcro nel concetto di *utilità attesa*. Secondo questo concetto l'individuo, se si comportasse in maniera razionale (mito bayesiano), dovrebbe ottenere il maggior vantaggio previsto, cioè scegliere tra le opzioni della situazione quella che garantisce la massima utilità. Questo si verifica sia nel caso di esiti incerti (scelte in condizioni di incertezza) sia nel caso di esiti certi (scelte in condizioni di certezza) Bernoulli (1738); von Neumann e Morgenstern (19479); Savage (1954).

[5] Le transazioni potenziali ed effettive avvenivano in modo decentralizzato in quanto gli studenti si muovevano liberamente all'interno della classe, ma le transazioni effettuate non erano contestualmente note a tutti. Pertanto non avvenivano conformemente all'ambiente istituzionale necessario a sottoporre a verifica la teoria della concorrenza perfetta.

simultaneamente ai partecipanti. Operando così Smith fu in grado di confermare la convergenza dei prezzi verso l'equilibrio. Anche i risultati ottenuti da V. Smith inizialmente non furono molto apprezzati, ma oggi tale filone è ancora oggetto di analisi e dibattiti.

Il secondo filone di studi sperimentali è stato sviluppato a partire dagli anni '50 e '60 da psicologi, sociologi e economisti interessati dalla teoria dei giochi.

Nel 1950 Dresher e Flood realizzarono un esperimento[6] che introdusse il gioco divenuto poi noto come *dilemma del prigioniero*.[7]

A due giocatori che non potevano dialogare tra loro, veniva presentata una matrice come quella riportata di seguito

	Colonna 1	**Colonna 2**
Riga 1	-1, 2	0.5, 1
Riga 2	0, 0.5	1, -1

Ciascuno poteva scegliere una riga o una colonna in ognuno dei cento turni in cui si svolgeva il gioco. Ad ognuna delle quattro possibili combinazioni di riga/colonna che i giocatori potevano scegliere, erano associate le ricompense, *payoff*, reali in moneta riportate nella tabella. Sommando le ricompense ottenute nei cento turni si poteva determinare la vincita totale di ciascun giocatore.

La teoria prevede che in un simile contesto l'equilibrio di Nash venga raggiunto con la combinazione riga due e colonna uno. La riga due è infatti sempre preferibile dal soggetto che sceglie rispetto alle righe in quanto garantisce il miglior risultato indipendentemente dalla scelta dell'avversario[8], idem per la colonna uno.

Dresher e Flood conclusero però che i soggetti tendevano a preferire un comportamento diverso da quello previsto dalla teoria, sebbene non optassero per un comportamento perfettamente cooperativo. Lo stesso Nash fornì una sua interpretazione dei risultati sottolineando come, secondo lui, la discrepanza tra teoria e risultati fosse dovuta ad una precisa serie di fattori: i giocatori interagivano per cento turni e quindi dovevano tenere conto delle reazioni dell'avversario. Secondo Nash i

[6] Tenutosi nel 1950 presso la Rand Corporation.
[7] Dai testi di Flood (1952), (1958).
[8] Zero contro -1 se la scelta è colonna uno, 1 contro 0.5 se la scelta è colonna due.

risultati sarebbero stati diversi se alcune condizioni fossero state diverse (scelte del partner rese note solo al termine dei cento turni, numero di turni indeterminato, coppie di giocatori diverse per ogni turno).

Sociologi e psicologi inizialmente non riuscivano ad accettare l'idea che gli individui potessero ragionare in modo da ottenere il peggiore risultato, o comunque un risultato non ottimale, ma con l'evolversi e il susseguirsi di numerosi esperimenti hanno dato origine ad una voluminosa letteratura sulla possibilità che si determinino comportamenti cooperativi tra i due "prigionieri".

Il terzo filone di analisi sperimentale riguarda il comportamento degli individui in situazioni semplici, ovvero in situazioni in cui è sufficiente ottimizzare e in cui non è necessario adottare strategie di comportamento. Gli esperimenti che appartengono a tale filone sono stati realizzati originariamente per verificare i postulati su cui si basa la teoria delle scelte in condizioni di incertezza formulata inizialmente da von Neumann e Morgenstern e in seguito da Savage. Tali studi hanno mostrato frequenti e sistematiche violazioni degli assiomi di base.

Oltre ai numerosi studi relativi ai problemi di formulazione delle decisioni nel campo dell'utilità attesa, altri esperimenti sono stati svolti per verificare la razionalità degli individui nel generare previsioni sui prezzi di mercato o per analizzare il comportamento degli individui nel caso di strategie ottimali di ricerca.

Oggi negli Stati Uniti l'economia sperimentale è ormai una disciplina riconosciuta e affermata; in Europa i paesi maggiormente all'avanguardia sono Inghilterra, Germania, Olanda, Francia e Italia. Proprio nel nostro Paese l''utilizzo della metodologia sperimentale si sta diffondendo rapidamente. Le aree maggiormente attive sono le Università di Alessandria, Milano, Pavia, Napoli, Roma, Siena, Torino, Trento, Bari e Catania, più alcuni Centri di ricerca tra i quali bisogna ricordare il CES (Centro di Economia Sperimentale) con sede presso la Luiss di Roma e il CEEL (Experimental Economic Laboratory) che risiede presso l'Università di Trento.

1.2 Esperimenti e simulazioni

Molto spesso i termini esperimento e simulazione vengono utilizzati come sinonimi e l'attività di sperimentazione viene confusa con quella di simulazione. In

realtà si tratta di due metodologie di analisi completamente diverse. La simulazione viene adottata per esplorare, generalmente tramite l'uso del calcolatore, le proprietà di modelli teorici, tipicamente quando essi si presentano in modo troppo complesso per essere risolti analiticamente. La sperimentazione serve invece ad osservare il comportamento effettivo dei soggetti. I due metodi possono però essere utilizzati congiuntamente e con proficui risultati: si ricorre alla sperimentazione per scoprire come si comportano gli agenti economici e si impiega successivamente la simulazione per esplorare le implicazioni di tali comportamenti.

Esperimenti e simulazioni possono però essere utilizzati anche parallelamente permettendo, come si vedrà in seguito, l'interazione in tempo reale di esseri umani e di agenti artificiali. Durante l'esperimento alcuni giocatori umani si confrontano con giocatori artificiali, costruiti secondo regole in grado di riprodurre e quindi simulare alcuni particolari atteggiamenti.

Le simulazioni intese come nuovo metodo di ricerca, sono anche in grado di automatizzare il processo che permette di derivare da una teoria le predizioni da sottoporre a verifica empirica. Nelle teorie tradizionali è lo scienziato che deriva dalla teoria le predizioni empiriche, essenziali per verificarne la correttezza. Nelle simulazioni, invece, tale compito non è più svolto dallo scienziato, bensì dal computer. Quando infatti la simulazione gira nel computer, i risultati che si ottengono sono le predizioni empiriche derivate dalla teoria stessa.

Le simulazioni possono essere usate anche prima della formulazione di una teoria, e più precisamente per elaborare le teorie stesse, per esplorarne e valutarne le caratteristiche e le implicazioni quando sono ancora in fase di costruzione. Nel caso in cui dalle prime simulazioni si possano rilevare evidenti errori che minano alla base la teoria stessa è possibile, senza ulteriori sprechi di tempo, correggere le proprie impostazioni senza dover attendere le smentite che deriverebbero dal successivo approccio empirico.

Il metodo degli "esperimenti mentali" (Parisi, 2001), marginalmente utilizzato nella scienza, viene automatizzato grazie alle simulazioni. La teoria e la simulazione vengono sviluppate congiuntamente nella mente dello scienziato e si ha un continuo flusso dalla testa alla simulazione. Utilizzando il metodo della simulazione lo scienziato continua ad elaborare le predizioni che dalla teoria derivano, ma facendo girare la

simulazione nel suo computer è in grado di verificare se le sue previsioni corrispondono ai risultati della simulazione. Si tratta di una vera e propria nuova forma di conferma empirica che non esiste nella scienza tradizionale. Il passo avanti che permette l'adozione del metodo simulativo è la possibilità di confrontare le predizioni di una teoria non solo con la realtà empirica, ma anche con i risultati ottenuti da una simulazione che incorpori la teoria stessa.

Nel paragrafo 1.3 si procederà con una breve analisi di alcune tipologie di esperimenti, mentre nel paragrafo 1.4 verranno analizzati alcuni vantaggi e svantaggi che si incontrano con l'uso delle simulazioni. Si è deciso di operare secondo questa linea per introdurre nel più chiaro e completo modo possibile la descrizione della parte applicativa della tesi che avrà il suo centro nel capitolo 6 in cui verrà descritta tutta la procedura con la quale sono stati progettati, strutturati e realizzati i giochi svoltisi il 3 maggio 2001 e il 19 aprile 2002 presso il laboratorio Liases della Facoltà di Economia di Torino.

1.3 Alcune tipologie di esperimenti, le caratteristiche principali

Un esperimento si compone fondamentalmente di una sessione in cui alcuni soggetti, siano essi esseri umani o agenti artificiali, vengono fatti interagire tra loro in base a delle istruzioni fornite da chi ha ideato l'esperimento. La gamma degli esperimenti che si possono effettuare è molto ampia, ma in base ad alcune particolari caratteristiche possono essere facilmente suddivisi in classi:

- *Gioco a giocatore singolo*

E' il caso più semplice: ogni singolo giocatore che partecipa all'esperimento non ha alcuna interazione con gli altri giocatori, effettua delle operazioni autonomamente e il programma si preoccupa di raccogliere le informazioni rilevanti sulle sue azioni.

Questo genere di gioco è molto utilizzato in psicologia, viene attuato tramite un questionario a cui ogni giocatore risponde individualmente. Le risposte vengono raccolte e costituiscono il database dell'esperimento.

- *Gioco a gruppi*

In questa seconda tipologia di esperimenti i giocatori vengono suddivisi in gruppi composti da un certo numero di persone. Lo svolgimento del gioco è vincolato all'interazione dei giocatori appartenenti a tali gruppi. L'interazione può essere fondamentalmente di due tipi: concorrenziale, e quindi lo scopo di ciascun giocatore è di realizzare un punteggio maggiore degli avversari, oppure collaborativa, in questo caso gli individui che appartengono allo stesso gruppo dovranno coordinarsi e unendo gli sforzi cercheranno di raggiungere l'obiettivo comune. Esiste poi ancora un'ulteriore forma di interazione, quella "anonima". I giocatori nello svolgersi dell'esperimento si influenzano vicendevolmente, ma non sono in questo caso consapevoli del tipo di interazione a cui partecipano.

I ruoli che ogni soggetto svolge all'interno di ogni singolo gruppo sono paritari o gerarchici. Nel primo caso ogni singolo individuo si trova nelle stessa condizione in cui si trovano gli altri elementi del gruppo, nel secondo caso, invece, è possibile avere una struttura gerarchica dei ruoli che i singoli soggetti hanno all'interno di vari gruppi.

Un elemento molto importante di ogni esperimento è la sua evoluzione temporale. Il gioco può essere strutturato in più turni, all'interno dei quali si ripete sempre la stessa situazione, oppure ad ogni turno è possibile che la situazione cambi, che le regole vengano modificate, che vari la composizione dei gruppi, i ruoli dei giocatori. Un ulteriore vincolo che può gravare sui partecipanti all'esperimento oltre al rispetto delle regole può essere un limite temporale entro cui operare (vengono dati dei tempi massimi entro cui effettuare le proprie scelte). Ogni gioco può essere ulteriormente classificato in base al tipo di evoluzione temporale delle azioni dei singoli giocatori in due differenti tipologie:

- *Gioco sincrono*

Il soggetto è vincolato temporalmente dalle azioni compiute dagli altri giocatori o dall'evoluzione del gioco. Le possibilità di compiere azioni può quindi venire inibita in attesa che si verifichi la condizione necessaria, come ad esempio il passaggio della mano in un qualsiasi gioco di carte. Una volta giocato, il giocatore deve attendere che

l'altro giocatore risponda alla sua mossa e nel frattempo non può eseguire un'altra azione.

- *Gioco asincrono*

Le azioni di ogni giocatore sono indipendenti dall'operato degli altri o dalla situazione del gioco. Un esempio di questo tipo di situazione è la simulazione di un mercato in cui i soggetti sono liberi di compiere azioni di compravendita senza essere vincolati nel tempo dalle azioni che gli altri componenti del mercato effettuano.

Se poi un esperimento è composto da diverse sessioni di gioco, vi possono essere sessioni sincrone, cioè che avvengono in maniera sequenziale, come situazioni di gioco in cui vi sono sessioni diverse che avvengono contemporaneamente.

Come si può intuire da quanto esposto, la gestione temporale di un esperimento è un aspetto basilare in quanto deve permettere di realizzare tutti i tipi di gioco sopra descritti, controllando le azioni di vari elementi e intervenendo nel caso in cui vi siano delle scadenze temporali prefissate.

Altre due caratteristiche di un esperimento che devono essere particolarmente curate per garantire la bontà e l'interpretabilità corretta dei risultati sono la raccolta e lo scambio dei dati e l'interfaccia grafica di fronte alla quale si troveranno i giocatori.

Durante lo svolgimento del gioco la comunicazione dei dati deve essere possibile sia dal giocatore all'entità di controllo del gioco, sia nel senso inverso[9], sia tra i giocatori stessi. Per quanto riguarda invece la necessità di raccolta dei dati, le tipologie di elementi che può essere utile annotare sono le stesse che vengono utilizzate durante lo svolgimento del gioco (una mossa, la quantità di merce venduta sono i dati che vengono scambiati durante il gioco e che vanno nello stesso momento registrati).[10]

I numerosi tipi di esperimenti che si possono realizzare richiedono l'utilizzo di un'interfaccia di tipo grafico verso il soggetto. L'interfaccia deve mostrare dati e oggetti di vario tipo al fine di consentirgli di prendere decisioni e di ottenere delle informazioni utili per lo svolgimento del gioco. E' quindi necessario un ampio set di componenti grafici che permettano la costruzione di interfacce composte. Può quindi essere necessario avere la possibilità di inserire dati numerici o testuali, caselle di scelta

[9] Caso in cui il tipo di esperimento preveda la ridistribuzione dei dati in qualche forma.
[10] Può essere necessaria una particolare attenzione ai dati di carattere temporale come ad esempio il tempo impiegato dai vari soggetti per effettuare la propria scelta.

tra le opzioni proposte, vari tipi di controlli come bottoni, tastiere numeriche, grafici di vario tipo, alberi di scelta, matrici di *payoff*. Elemento importantissimo al fine di evitare distorsioni dei risultati è che l'interfaccia fornisca al soggetto tutti gli strumenti necessari per lo svolgimento del gioco in modo che il giocatore non disponga di materiale personale di cui non si ha il controllo. Nel caso in cui il soggetto sottoposto all'esperimento decida di utilizzare gli strumenti forniti dallo sperimentatore è necessario registrare tutte le operazioni al fine di poterle considerare durante l'analisi dei dati.

1.4 L'uso delle simulazioni: vantaggi e svantaggi

Le teorie espresse in forma tradizionale con parole, simboli matematici, schemi grafici si limitano a spiegare la realtà, mentre le teorie espresse come simulazioni sono in grado di riprodurre la realtà stessa ricreandola nel computer. Ma le simulazioni, oltre che un potente e nuovo mezzo utilizzato per esprimere le teorie nella scienza, sono anche veri e propri laboratori virtuali. Così come avviene in un laboratorio reale, i risultati di una simulazione, fenomeni virtuali che lo scienziato osserva quando la simulazione gira nel computer, si verificano in condizioni che possono essere facilmente controllate. Lo sperimentatore è in grado di manipolare le condizioni che determinano o influenzano il verificarsi dei fenomeni studiati, può modificare parti della simulazione, interferire sulle variabili oggetto dello studio, variare il valore quantitativo dei singoli parameri, ma soprattutto può valutare in tempo reale gli effetti di tali modifiche.

La maggiore novità per la scienza, introdotta dalle simulazioni, è il fatto che esse non siano solo teorie, ma anche realtà[11]. Si tratta ovviamente di realtà artificiale, ovvero di realtà ricostruita dallo scienziato. Se infatti consideriamo i criteri con i quali D. Parisi (2001) definisce la realtà scopriamo come tali criteri si adattino perfettamente anche alle simulazioni. La realtà è i) quello cui abbiamo accesso con i nostri sensi (vista e udito, ma anche i sensi che ci fanno percepire i movimenti del nostro corpo, un dolore fisico, ecc.); ii) quello su cui possiamo agire e che reagisce alle nostre azioni; iii) quello

[11] Domenico Parisi in "Simulazioni, la realtà rifatta al computer"

che vincola, ma nello stesso tempo ci consente di svolgere le nostre azioni. La realtà artificiale, di conseguenza, esiste dal momento che esistono gli esseri umani[12]. Dal momento che una simulazione può essere osservata sul terminale di un calcolatore, che si può agire su di essa modificando ad esempio parti di codice o semplicemente interagendo con l'interfaccia utente e ottenendo risposte alle nostre azioni, e che mentre da un lato ci vincola, consentendoci di compiere alcune azioni ma vietandocene altre, dall'altro costituisce il mezzo per svolgere determinate azioni, allora chi costruisce una simulazione sta di fatto creando una realtà artificiale.

Le simulazioni sono in grado di fondere insieme la costruzione di modelli della realtà e l'elaborazione di teorie matematiche in grado di cogliere la struttura quantitativa della realtà, ma il fatto che abbiano come cuore un calcolatore rende possibile sviluppare modelli molto complicati e dinamici, composti di meccanismi che operano, di processi che si generano e di risultati che vengono prodotti. Inoltre, l'aver espresso una teoria o un modello nella forma di una simulazione garantisce, per definizione, l'assenza in tale teoria/modello di concetti vaghi, parti poco chiare o mancanti, parti non sufficientemente dettagliate o confuse; infatti per poter tradurre una teoria in un programma per computer è indispensabile che la teoria non sia intaccata da questo genere di imprecisioni, altrimenti la simulazione non funzionerebbe oppure pur funzionando produrrebbe risultati evidentemente distorti.

Addentrandosi poi nel campo economico, i maggiori vantaggi che si possono ottenere con l'uso delle simulazioni e quindi con lo studio sperimentale dei principi economici sono la replicabilità e il controllo. Con il termine replicabilità si intende la possibilità per altri ricercatori di riprodurre un esperimento verificandone i risultati in modo indipendente. Se si studiano dei dati naturali i processi che li producono sono invece unici per quanto riguarda la loro ubicazione spaziale, temporale ed avvengono in ambienti in cui altri fattori inosservati e spesso inosservabili sono in continuo cambiamento. Nel laboratorio virtuale costituito dalla simulazione è invece possibile creare procedure che controllino gli effetti della distinzione spazio-temporale degli esperimenti consentendo una duplicazione dell'ambiente in cui si conduce il test. Poter ripetere l'esperimento nelle stesse ed identiche condizioni per più sessioni successive garantisce la validità e verificabilità dei dati ottenuti. I costi di raccolta e verifica dei

[12] La realtà naturale esiste indipendentemente dall'esistenza degli esseri umani ed ha caratteristiche che non sono la conseguenza delle azioni degli uomini.

dati sono molto maggiori nel caso in cui sia posto sotto analisi un esperimento naturale, infatti il reperimento di tali dati è generalmente molto più difficoltoso che in un ambiente controllato e limitato come quello di una simulazione.

Per controllo si intende invece la capacità di manipolare le condizioni di laboratorio in modo da poter utilizzare i comportamenti osservati per la valutazione di teorie e politiche alternative. Anche l'analisi di dati reali potrebbe teoricamente essere possibile, ma con molto maggiori difficoltà derivanti dall'impossibilità di raccogliere ed aggregare dati reali in modo assolutamente preciso per poterli confrontare con diverse teorie. In alcuni casi, poi, potrebbe addirittura non esistere una situazione economica reale che corrisponda alla teoria in analisi. Sarebbe quindi necessaria l'introduzione di innumerevoli ipotesi ausiliarie che possono compromettere la significatività dei test effettuati sulle ipotesi principali, minando alla base l'intero esperimento. Il ricercatore che riproduce nel suo laboratorio virtuale le situazioni naturali delle teorie economiche oggetto dello studio è in grado di conoscere e controllare tutte le caratteristiche (condizioni ed ambiente) eliminando quindi la maggior parte delle condizioni ausiliarie che possono compromettere la validità e la veridicità dei risultati ottenuti.

Le simulazioni e di conseguenza gli esperimenti svolti con l'aiuto delle simulazioni, oggetto di questa tesi, se da un lato propongono interessanti vantaggi e prospettive per lo studio dell'economia, dall'altro possono essere oggetto di critica. Una delle principali osservazioni che viene sollevata nei confronti di simulazioni e di questa tipologia di esperimenti in genere è l'eccessiva semplificazione che tali strumenti richiedono rispetto alle vere situazioni (ad esempio di mercato) che si presentano nella realtà. Questa critica, però, si fonda su una mancata comprensione del significato profondo del termine simulazione. Una simulazione è innanzitutto una teoria (che viene espressa sotto forma di programma per il computer) e, come tutte le teorie scientifiche, deve semplificare. Le teorie sono utili proprio perché in grado di semplificare e quindi di consentire allo studioso di cogliere l'essenziale che generalmente sta al di sotto dei complicati e vari fenomeni osservati. Il vero problema è invece di stabilire se la simulazione adotta le semplificazioni giuste, ovvero se ciò che resta al di fuori della simulazione è effettivamente irrilevante ai fini della comprensione della realtà osservata. Più corretta e fondata è invece la critica che si espone verso la simulazione

che tende a riprodurre una realtà effettivamente troppo semplificata. E' infatti ovvio che si desideri una riproduzione dei fenomeni oggetto di studio il più fedele possibile. Mentre però, le teorie espresse in modo tradizionale non potranno evolversi e resteranno necessariamente semplici, è realistico aspettarsi che le simulazione si evolvano includendo sempre maggiori e più realistiche caratteristiche . Grazie proprio alle caratteristiche di alta capacità di calcolo insite nel cuore delle simulazioni (il computer) è possibile rendere i fenomeni simulati sempre più simili a quelli reali senza che i limiti della mente umana facciano da ostacolo alla scienza. Non va poi dimenticato il fatto che il metodo sperimentale fornisce uno strumento di verifica importantissimo in quanto se una teoria non trova la sua verifica in ambiente simulato probabilmente non è nemmeno applicabile ad una situazione reale; è così possibile modificare immediatamente la teoria senza ulteriori sprechi di tempo e risorse.

Il metodo della simulazione va al cuore di quello che è il problema principale di tutte le scienze dell'uomo ed in particolare delle scienze sociali tra cui è compresa ovviamente l'economia, ovvero l'insufficiente integrazione e interazione tra teorie e fatti empirici. In generale nelle scienze dell'uomo o le teorie mancano del tutto, o dalle teorie è impossibile derivare predizioni specifiche da confrontare con le evidenze empiriche. Su queste basi è evidentemente impossibile costruire una scienza matura. Proprio con l'adozione del metodo simulativo è possibile cambiare questo stato di cose. Poiché le simulazioni sono teorie, se una scienza lavora con le simulazioni allora questa scienza necessariamente lavora su teorie. Inoltre queste teorie espresse sotto forma di simulazioni sono esplicite, precise e dettagliate (altrimenti non potrebbero essere incorporate in un programma che giri efficientemente sul computer). Le tecniche sperimentali divengono quindi centrali nell'approccio cognitivo allo studio dell'economia.

Grazie a questo nuovo approccio è possibile rifiutare l'idea di agente massimizzante e di razionalità "olimpica". L'insoddisfazione per le capacità interpretative dei modelli tradizionali genera questa necessità di ricerca di un percorso alternativo che è resa possibile proprio grazie all'introduzione delle simulazioni. I diversi lavori sperimentali già effettuati hanno posto in discussione gli assiomi di base della teoria neoclassica dimostrando che gli individui non agiscono in base a regole e criteri ottimali. Le regole che ciascun individuo implicitamente adotta dipendono dai

particolari processi di apprendimento messi in atto dai soggetti stessi. Le regole non sono fisse, ma evolvono continuamente[13]. I processi di apprendimento rivestono quindi un'importanza centrale per la comprensione dei comportamenti individuali e organizzativi.

L'economia sperimentale e l'utilizzo delle simulazioni in particolare, rappresentano uno degli strumenti empirici, di analisi necessari per lo sviluppo di questo nuovo settore di studi[14] che si dedica all'indagine dei processi cognitivi e decisionali.

[13] Si tratta di un microapprendimento continuo.

[14] Simon (1999): "The traditional empirical tool of economics, collection of aggregated data and their analysis by statistical regression, can only provide one weapon in the armony, and that not the most important…

…This means that economists must be trained in the methods of observing and interviewing, of taking and analysing verbal think-aloud protocols, of extracting information about decision processes from written records…

…A second development… is trainig research economists in the design and conduct of laboratory experiments, and in the use of experimental data to discover and test economic theories…"

PARTE SECONDA

Altruismo, egoismo, cooperazione e apprendimento

2.1 Altruismo ed egoismo

Al fine di comprendere a fondo gli obiettivi e i risultati degli esperimenti svolti con l'aiuto delle simulazioni oggetto di analisi di questa tesi, è indispensabile studiare un po' più approfonditamente i concetti di altruismo ed egoismo.

Ad accrescere la complessa analisi di tali argomenti interviene l'interrelazione che questi temi presentano con la teoria della razionalità e delle emozioni, con concetti quali coordinamento, apprendimento e istituzioni.

Poiché come vedremo nel successivo capitolo, oggetto dei due esperimenti sarà l'interazione tra agenti umani da un lato, e artificiali "altruisti" e "egoisti" dall'altro, è necessario innanzitutto definire che cosa si intende, nella letteratura economica, per altruismo ed egoismo.

In prima istanza per altruismo possiamo intendere un comportamento non immediatamente e direttamente rivolto al puro interesse personale; nell'ampia letteratura al riguardo vengono spesso utilizzati termini quali *kindness, fairness, loyalty, cooperation* a seconda dell'ambito e del contesto in questione. Certamente il vocabolo altruismo è quello che più naturalmente si contrappone al termine egoismo. La teoria economica standard definisce egoista un individuo motivato esclusivamente dalla propria funzione di utilità. Si potrebbe obiettare tale affermazione o se non altro tentare di farla vacillare includendo nella funzione di utilità di un individuo anche il benessere o il livello di consumo di altri individui. Dato che l'obiettivo finale dell'*homo oeconomicus* in questione è comunque la massimizzazione della propria funzione di utilità, non avrebbe più senso parlare di agenti altruisti in quanto, data l'ipotesi, nessun comportamento potrebbe essere realmente altruista. Come però rilevato da Sacco e Zamagni (1994), la precedente definizione di comportamento egoistico è solamente una definizione tautologica in quanto onnicomprensiva: include infatti qualsiasi tipo di comportamento, sia esso altruistico o egoistico. Rimane invece la distinzione tra i due comportamenti se ci si limita a far riferimento ai vantaggi indiretti che un individuo egoista può ottenere se nella propria funzione di utilità tiene conto del benessere degli

altri individui. In questo caso, anche ponendosi in una prospettiva di selezione naturale, è possibile giustificare la sopravvivenza dei geni dell'altruismo.

Come già appare da queste prime considerazioni, siamo in presenza di un forte legame tra le idee di altruismo e di razionalità. Spesso infatti la letteratura si è posta il problema-obiettivo di giustificare l'altruismo in termini di razionalità. Alla base di questo tentativo sta l'osservazione empirica di atteggiamenti altruistici. Già gli esperimenti di Dresher e Flood del 1950, relativi all'applicazione della teoria dei giochi, mostravano infatti come nel caso di ripetizione di situazioni del tipo dilemma del prigioniero emergesse una evidente tendenza alla cooperazione. Anche esempi di comportamenti reali analizzati e studiati da Andreoni (1995), Croson (1996), Rabin (1993), Hudson e Jones (1994) mostrano come l'ipotesi, per lungo tempo presa come un dato di fatto di assoluta propensione all'egoismo, non sia pienamente corretta.

Considerato ciò, Sacco e Zamagni (1994) mostrano però come l'introduzione dell'altruismo nello schema economico standard consenta di spiegare ed interpretare solo un'esigua parte dei comportamenti non egoistici osservabili.

Se invece della classica visione della razionalità se ne adotta una alternativa che permetta anche di rimanere al di fuori dell'ottica di massimizzazione, diventa più facile giustificare i comportamenti altruistici superando alcuni dei molteplici limiti logici e metodologici altrimenti presenti. Anche in questa "nuova" prospettiva, l'altruismo continua ad essere un comportamento vantaggioso per chi lo attua, anche se ovviamente in termini diversi.

2.2 Homo Oeconomicus e altruismo

Come già anticipato nel paragrafo precedente, secondo l'approccio economico standard il comportamento considerato ad ogni effetto naturale è quello egoista. Hahneman, Knetsch e Thaler (1986) hanno osservato che nella letteratura economica tale ipotesi ha assunto forme differenti. In alcuni casi si tratta di ipotesi *as if* ed hanno natura prevalentemente metodologica. D'altro canto c'è chi nega anche da un punto di vista sostanziale la possibilità di qualsiasi comportamento altruistico. Sen (1985) distingue fondamentalmente tre aspetti di "*privateness*" nel comportamento ipotizzato dalla teoria economica standard e dalla teoria dei giochi.

- L'individuo è prima di tutto *self-centered welfare*, il suo benessere è legato esclusivamente al proprio consumo e non prevede simpatie o antipatie per gli altri.
- Ciascun individuo ha come obiettivo la massimizzazione del valore atteso del proprio benessere e non è interessato a quello degli altri (*self- welfare goal*).
- Le scelte compiute sono mirate al puro perseguimento dei propri obiettivi senza vincoli di riconoscimento dei fini perseguiti dalle altre persone (*self-goal choice*).

Data tale fondamentale - almeno per la nostra analisi - definizione, i comportamenti altruistici sono quelli che deviano, sia pure solo parzialmente, da anche solo una di queste ipotesi.

Le tre più consuete definizioni di altruismo che si possono incontrare nella letteratura economica sono invece le seguenti:

- Secondo la teoria dei giochi un agente razionale coinvolto in una situazione tipo dilemma del prigioniero sceglierà di defezionare (ovvero confessare). Di conseguenza Bergstrom e Stark (1993) definiscono altruista chi decide di cooperare.[10]

- Becker (1976) definisce altruista l'individuo la cui funzione di utilità include il consumo degli altri (in coerenza con quanto previsto dalla teoria neoclassica).

- Arrow (1962 e 1986) e Pauly (1968) rilevano invece come in base alla teoria economica tradizionale, possano insorgere problemi di opportunismo, il tipico *moral hazard*, in situazioni di asimmetria informativa. Un tipico esempio è il caso datore di lavoro e dipendenti. Anche in questo caso siamo di fronte ad un dualismo concettuale: da un lato c'è chi si preoccupa di analizzare come si possano incentivare i lavoratori ad essere altamente produttivi, dall'altro lato chi sostiene che il rapporto tra impresa e lavoratore possa essere basato su lealtà e collaborazione. Entrambe le teorie conducono comunque ad una visione della razionalità e del comportamento individuale alternativa a quella neoclassica.

[10] Anche negli esperimenti descritti in Andreoni e Miller (1993), Rabin (1993), Andreoni (1995) è utilizzata una definizione riconducibile a quella descritta.

2.3 Altruismo e razionalità

Una delle ipotesi che tenteremo di testare, grazie ai due esperimenti svolti, sarà valutare se un comportamento altruistico, unito ad un a certa dose di collaborazione, permetterà di ottenere un vantaggio (nel caso in questione un punteggio) maggiore rispetto ad un atteggiamento egoistico apparentemente più "remunerativo". Già Becker, Rotemberg e altri studiosi hanno proposto modelli nei quali sembra che la selezione naturale da sola non permetta a un comportamento egoistico di sopravvivere.

Il modello studiato da Becker nel 1976 prevede un mondo composto da due soli individui, uno egoista ed uno altruista, che, come visto in precedenza, include nella sua funzione di utilità anche il consumo dell'altro soggetto. Ovviamente le scelte di produzione e consumo compiute dai due agenti risultano modificate rispetto alla situazione in cui ci siano solo persone egoiste. L'individuo altruista può ora scegliere di trasferire una parte della sua ricchezza all'altro agente e tale trasferimento può continuare fino a quando l'utilità marginale del consumo dell'altruista è uguale a quella dell'egoista. Anche nelle decisioni relative alla produzione l'individuo altruista evita di realizzare operazioni che creano extrareddito per lui a scapito della concorrenza. Tale atteggiamento ha implicazioni importantissime: anche l'individuo egoista prenderà in considerazione la ricchezza totale prodotta evitando di danneggiare l'altro individuo se tale danno si rifletterà sui trasferimenti che riceve. Il "mini mondo" studiato da Becker può essere ampliato ed esteso ad un mondo pluriagente

Rotemberg (1994) propone un modello in cui gli individui sono dotati di due funzioni di utilità: quella classica, più una che tiene conto degli altri individui. Gli agenti che interagiscono in questo mondo possono scegliere razionalmente un determinato livello di altruismo, decidendo di far dipendere la loro utilità anche da quella degli altri. Massimizzare la funzione che deriva dall'unione delle due precedenti consente di ottenere livelli di utilità altrimenti non raggiungibili.

Becker, sempre nel 1976, estende il suo modello alla situazione in cui l'utilità dipende dalla probabilità di riprodursi (*fitness* genetico). Ancora una volta l'altruismo appare

evidentemente più efficiente dell'egoismo, risultando la "strategia" vincente nel processo di selezione naturale.

Bergstrom e Stark (1993) propongono un modello in cui, ancora una volta, un agente egoista imitatore, e cioè un individuo che imita i genitori o i vicini, sceglie razionalmente un comportamento altruista che gli permette di raggiungere un *payoff* molto più alto.

Da questa analisi si evince come, in ottica evoluzionista, il concetto di altruismo indica un comportamento che riducendo il benessere di un individuo incrementa la fitness degli altri.

In conclusione Simon (1983 e 1993) individua due definizioni di altruismo, una forte, che implica il sacrificio della propria "capacità di adattamento" a vantaggio degli altri e una debole, che è invece interpretabile come interesse personale e che è l'unica sostenibile nell'ambito di una selezione naturale.

2.4 Altruismo ed evoluzione biologica

L'idea che le preferenze degli individui non siano frutto dell'arbitrarietà, ma siano piuttosto il prodotto dei processi evolutivi è la pietra miliare della sociobiologia (Wilson, 1975). Così la tendenza alla cooperazione potrebbe essere innata e connaturata alla stessa natura umana essendo il risultato di complessi processi evolutivi, come confermano anche gli studi degli psicologi evolutivi Cosmides e Tobby (1992). Questi ultimi hanno per esempio osservato come una particolare specie di pipistrelli vampiri abbia l'abitudine di cedere parte del sangue predato ai simili che non sono riusciti a procurarsene con la caccia. La biologia è ricca di teorie che spiegano tali fenomeni che clamorosamente dimostrano come in natura una qualche forma di altruismo sia presente anche negli animali. La *group selection* è una di queste. Secondo questo filone di studi, i gruppi composti da individui altruisti hanno maggiori possibilità e maggiori capacità di sopravvivenza, espandendosi così a scapito dei gruppi caratterizzati dal prevalere di elementi egoisti. I cambiamenti evolutivi sarebbero così determinati dai differenti tassi di estinzione dei gruppi piuttosto che degli individui stessi. Questo filone di studi è stato molto criticato in quanto la selezione a livello di gruppo è impossibile dato che l'unità elementare di ogni gruppo, colui che si riproduce, è l'individuo stesso.

La *kind selection* (selezione della razza) è un'altra delle teorie sopra citate. L'altruismo sarebbe così rivolto quasi esclusivamente ai propri parenti. Così anche il sacrificio di un individuo viene visto come un bene in quanto permetterebbe ai geni presenti nella famiglia di sopravvivere e di propagarsi (Hamilton 1964). L'*inclusive fitness*, riproduzione dei propri geni, è secondo questa tesi l'obiettivo delle scelte compiute dagli individui e dei loro eventuali sacrifici; la discendenza diretta passa così in secondo piano.[11]

Alcuni scienziati evoluzionistici (ad esempio Alexander, 1977 ed Hoffman, 1981) ritengono che in realtà il principio della *kind selection* non richieda necessariamente la parentela tra gli individui (infatti gli stessi pipistrelli vampiri cedono il sangue anche ai non consanguinei) e che sia sufficiente che i soggetti pensino di essere parenti. Allora se si pensa che in una fase primordiale era possibile dedurre la parentela con un individuo quasi solo in base all'appartenenza allo stesso gruppo e quindi al livello di somiglianza, è possibile che la tendenza ad aiutare le persone simili sia diventata quasi naturale ed ormai indipendente dalla parentela (Krebs, 1982).

Un ulteriore filone di pensiero è invece centrato sul concetto di altruismo reciproco, o scambio differito di favori. In questo caso lo svantaggio che conseguirebbe oggi un egoista nel compiere un gesto altruista sarebbe ricompensato domani dal vantaggio di beneficiare del comportamento favorevole degli altri individui. Ovviamente anche in questo caso ci possono essere dei problemi: poiché lo scambio di benefici non è immediato, c'è il rischio che alcuni elementi non ricambino il favore ricevuto. Diventa così indispensabile la creazione di meccanismi che "eliminino" chi non coopera (i pipistrelli tendono infatti a cedere parte del sangue a compagni dai quali abbiano ricevuto tale favore).

Tutte queste teorie inducono a rilevare meccanismi che sembrano effettivamente esistere ed è possibile modellare il precedente schema con la strategia del *tit for tat* e applicarlo alla teoria dei giochi. Cosmisdes e Tobby (1992) hanno effettuato un'ampia serie di esperimenti che mostrano nettamente come gli individui abbiano sviluppato meccanismi cognitivi che permettono loro di individuare i comportamenti egoistici e di escludere dai loro gruppi chi li attua.

---

[11] Questa spiegazione è una sorta di risposta all'ipotesi di gene egoista introdotta da Dawkins nel 1976. Secondo lo studioso, i geni essendo egoisti sono esclusivamente interessati a sopravvivere e riprodursi e, se necessario, tutto ciò anche a scapito dell'organismo che li ospita.

2.5 Altruismo, evoluzione culturale e norme sociali

In alternativa alla selezione genetica, ma sempre restando nell'ambito di un'analisi evoluzionistica, è possibile spiegare l'altruismo facendo ricorso all'idea di selezione culturale delle norme. Secondo quest'ottica le norme culturali sono gli strumenti che garantiscono comportamenti altruistici nell'interazione sociale, anche se gli individui fossero per loro natura (geneticamente) egoisti. Proprio in questo caso sarebbero necessarie adeguate norme culturali.

Secondo gli studi condotti da Hayek (1967), e in accordo con Smith, gli individui avrebbero una tendenza innata ad agire secondo le regole del contesto sociale in cui vivono ed operano, anche se il motore dell'economia è l'interesse personale. Le regole vengono così selezionate in base all'efficienza per il gruppo. Ecco che la norma[12] diventa elemento fondamentale per un gruppo sociale al fine di garantire l'indispensabile cooperazione. "Le proprietà individuali importanti per l'esistenza e la preservazione del gruppo, e di conseguenza anche per l'esistenza e la preservazione degli stessi individui, hanno preso forma tramite la selezione di quelle provenienti dagli individui viventi in gruppi che a ogni stadio dell'evoluzione hanno mostrato la tendenza ad agire in base a regole tali che hanno reso il gruppo più efficiente" (Hayek, 1967, 153). Secondo lo studioso austriaco le regole di condotta possono essere innate o apprese, e i due tipi di regole possono interagire.

Proprio il mercato, tra le molteplici istituzioni, è lo strumento che consente ad individui egoisti e che non si conoscono di collaborare e cooperare per raggiungere il reciproco benessere. Certamente un'istituzione così complessa non sempre è in grado di funzionare correttamente e diventano indispensabili altre istituzioni e delle norme culturali, tra le quali la reputazione è l'esempio più lampante. Secondo Hayek essa è necessaria a garantire il corretto funzionamento del mercato.

[12]" Col termine di norma si intende come l'individuo o il gruppo dovrebbe comportarsi nelle più svariate situazioni, la prescrizione di come un individuo o un gruppo sociale dovrebbe o non dovrebbe pensare, sentire o agire in determinate situazioni" (F. Demarchi e A. Ellena, 1976).

Anche Tullock (1985 e 1999) sottolinea l'essenzialità della reputazione. Proprio l'interesse a crearsi una fama di persone corrette spinge gli individui, che magari vorrebbero defezionare comportandosi scorrettamente nei confronti degli altri, a comportarsi in modo da massimizzare l'utilità collettiva.

Ancora, secondo Hayek, è essenziale evidenziare il legame tra i meccanismi sociali e quelli biologici. Secondo lo studioso, e in linea con quanto evidenziato anche da Simon nel 1983 e nel 1993, esiste una naturale tendenza delle persone ad imitare ed apprendere dagli altri individui le norme sociali che hanno natura culturale. Gli individui essendo per natura docili e tendendo a dipendere dai consigli altrui, si ritrovano ad obbedire anche ad alcune regole che in realtà vanno a loro svantaggio. Complessivamente però la docilità è vantaggiosa poiché i costi sono minori dei guadagni.

Anche Bicchieri (1990), Jeffrey e Skyrms (1997) ottengono la cooperazione come risultato dei processi dinamici che portano a definire norme sociali.

2.6 Il tit for tat

"The evolution of cooperation" è il lavoro con cui Axelrod nel 1984 analizza la regola del *tit for tat*. In situazioni di interazione ed incertezza risulta essere il principale strumento utilizzabile per modellare l'evoluzione, sia essa culturale o biologica, di strategie non egoistiche. Axelrod nel suo lavoro propone un'ampia dimostrazione logica della possibile diffusione a livello sociale di un comportamento alternativo al meno efficiente egoismo puro: la cooperazione. Il modello studiato dall'economista permette di rappresentare l'apprendimento individuale o collettivo sia a livello culturale che genetico. Per far emergere le sue teorie Axelrod invitò una serie di studiosi a proporre algoritmi per un torneo basato su una serie di iterazioni del dilemma del prigioniero. Furono presentate e poste a confronto, utilizzando simulazioni al computer, moltissime strategie. La popolazione era quindi composta da una serie molto ampia di algoritmi diversi. Una selezione naturale basata sui *payoff* ottenuti dalle varie strategie escludeva turno dopo turno quelle meno efficienti, permettendo a quelle più efficaci di svilupparsi e diffondersi. La strategia vincente risultò essere appunto quella del *tit for tat*, ovvero coopera se il tuo avversario ha cooperato nel turno precedente, altrimenti

defeziona. L'altruismo risulta così essere potenzialmente vantaggioso per chi lo attua ed essere comunque legato al comportamento dell'altro, che viene imitato, e alla reciprocità di trattamento.

Skyrms (1996) presenta un esercizio concettualmente simile a quello di Axelrod. Utilizzando una serie di simulazioni, egli analizza come i differenti comportamenti altruistici possano diffondersi a livello sociale. Anche Skyrms parte da un semplice gioco: una versione semplificata dell'*ultimatum game* in cui due individui devono spartirsi una torta[13]. A certe condizioni, strategie che prevedono forme di altruismo possono risultare vincenti e diffondersi a livello sociale. Skyrms propone una serie di applicazioni a livello di selezione biologica e culturale dei comportamenti.

Un'estensione del modello si basa sull'assunzione che gli accoppiamenti non siano casuali, ma che siano il risultato di un qualche meccanismo di correlazione. Così individui con la stessa strategia tenderanno a giocare tra di loro più frequentemente rispetto a coloro i quali adottano strategie differenti. E proprio tale meccanismo di correlazione è per certi aspetti paragonabile a quanto previsto dalla strategia del *tit for tat* ed è anche molto simile al processo di selezione dei partner, legato al desiderio di crearsi una buona fama, come evidenziato da Tullock.

2.7 Incertezza, razionalità limitata e altruismo

Abbiamo già accennato nei precedenti paragrafi al ruolo dell'incertezza nel determinare comportamenti altruistici. Gli individui dei modelli ad agenti, temendo per l'incertezza sull'evolversi della mondo in cui interagiscono, hanno sviluppato una "preferenza per i tipi di azioni le cui conseguenze siano prevedibili e paura nei confronti di quei tipi di azioni le cui conseguenze siano imprevedibili." (Hayek 1967). Non possedendo una conoscenza perfetta, gli agenti devono adeguarsi e seguire le norme che provengono dall'ambiente in cui operano.

Secondo Witt (1998) a causa della razionalità limitata, gli individui leggono la realtà attraverso gli schemi di cui dispongono in base alla propria esperienza. Non sono

[13] L'*ultimatum game* prevede due giocatori, uno in posizione attiva ed uno passiva. Il giocatore attivo propone una divisione di una data somma fra lui e l'altro giocatore. Se questi accetta la divisione proposta, entrambi ottengono la rispettiva quota sulla quale hanno raggiunto l'accordo. Se il giocatore passivo rifiuta, nessuno ottiene nulla.

quindi in grado di immaginare tutte le possibile alternative rispetto a qualsiasi problema. Poiché lo sviluppo della conoscenza avviene generalmente all'interno di ambienti sociali (quale ad esempio l'impresa), l'interazione con gli altri, l'imitazione e la comunicazione portano alla creazione di schemi interpretativi condivisi.

Come emerge dai lavori sull'analisi dei giochi ad equilibri multipli di Schelling (1958 e 1969)[14], l'incertezza risulta correlata alla nascita ed allo svilupparsi di norme e comportamenti sociali. Emerge così come nei giochi di coordinamento i soggetti scelgono i punti focali che garantiscono la minore incertezza sul risultato.

Anche Shafir e Tversky (1992) utilizzano i concetti di incertezza e razionalità limitata, anche se talvolta con differenti implicazioni, per spiegare i comportamenti cooperativi. I due studiosi confrontano i tassi di cooperazione in un esperimento del tipo dilemma del prigioniero ad un turno unico con i tassi di cooperazione in un analogo esperimento in cui ai soggetti viene comunicata in precedenza la scelta compiuta dal rivale. Nel secondo caso è possibile osservare tassi di cooperazione minori. Gli individui cooperano meno anche quando viene loro detto che il partner ha scelto di collaborare. Anche in questo caso l'incertezza gioca un ruolo dominante.

Il modello di Rawls (1971) evidenzia come i soggetti siano disposti ad accettare un tipo di società che tenda a ridurre le disuguaglianze, non sapendo a priori quale posizione occuperebbero all'interno di essa.

Secondo Sen (1985) un comportamento altruistico nasce da come la persona vede se stessa in rapporto agli altri. Così il senso di identità può spiegare la scelta di cooperare anche nell'ambito del dilemma del prigioniero in quanto spingerà l'individuo a non ragionare in termini di obiettivi personali e ad accettare regole di condotta utili per il gruppo. Secondo queste teorie dunque la cooperazione non nascerebbe dall'incertezza, ma al contrario dall'esistenza di una conoscenza comune.

[14] Negli esperimenti in esame, i soggetti dovendo dividersi in maniera concorde una cifra pari a 100 senza comunicare, sceglievano quote omogenee del 50% a testa. Questa scelta non era dettata principalmente da motivi di equità o altruismo ma dal fatto che 50% rappresenta un *punto focale*, cui entrambi i soggetti giungono senza comunicare. Infatti se i soggetti devono spartirsi una cifra indicando una stessa sequenza di lettere, ciascuna delle quali individua un giocatore, scelgono di elencare le lettere nella maniera più intuitiva anche se questo porterà a compensi diversi (Schelling, 1958 e 1969).

2.8 Il ruolo dell'apprendimento

Come vedremo nel capitolo relativo agli esperimenti svolti a Torino ed Alessandria, l'apprendimento gioca un ruolo fondamentale, soprattutto nell'ambito dell'economia sperimentale in cui si vogliano far interagire persone ed agenti artificiali e in cui si desideri valutare se tali agenti sono in grado di influenzare in un qualche modo il comportamento degli umani. Tentiamo dunque ora di fare un rapido excursus relativo alla letteratura economica per comprendere come tale argomento viene trattato.

Hayek (1963) è un sostenitore della centralità dell'apprendimento. Lo studioso austriaco pone una particolare enfasi sulla capacità degli individui di "percepire regole (o regolarità o schemi) nell'azione degli altri". Proprio l'imitazione consente agli individui di fare propria la gerarchia di valori tipica del gruppo al quale appartengono.

Secondo Witt (1985) l'altruismo è in larga parte frutto di eredità culturali e dipende inoltre dall'esperienza personale. Poiché le preferenze per l'altruismo sono legate all'apprendimento esse sono in grado di evolvere e cambiare in continuazione.

Anche Simon ritiene centrali i processi di apprendimento. Gli individui tendono ad affidarsi alla conoscenza comune disponibile nell'ambiente in cui interagiscono. Sia Witt che Simon propongono spunti per l'analisi dell'impresa alternativa a quella proposta da Ross (1973) con alla base il modello principale-agente. Witt, in particolare, critica la visione dell'agente opportunista. I processi di apprendimento che avvengono all'interno dell'impresa possono generare schemi che inducono al coordinamento e all'altruismo. Secondo Simon anche lealtà e identificazione nell'impresa possono essere elementi rilevanti nella fondamentale motivazione dei lavoratori.

2.9 Altruismo ed emozioni

Molto spesso si tende a dare per scontato il ruolo del legame esistente tra altruismo ed emozioni e teoricamente le emozioni determinerebbero il comportamento altruistico. Diventa allora inevitabile soffermarsi, anche solo brevemente, sulla letteratura psicologica.

L'indagine in campo psicologico mira spesso a tentare di comprendere se effettivamente empatia[15], simpatia e altruismo siano collegati, come sostengono Eisemberg e Miller (1987) e come confermato dai molteplici lavori di natura empirica. Secondo quanto studiato da Hoffman (1981), l'empatia è lo stato d'animo innato di chi soccorre chi si trova in una situazione di pericolo. Non si tratterebbe dunque di un elemento legato a particolari culture, bensì sarebbe frutto della selezione naturale.

Aronfreed nel 1970 osservava come un bambino acquisisca l'empatia collegando i propri sentimenti di piacere o di dolore alle corrispondenti emozioni degli altri tramite un processo di condizionamento. I sentimenti degli adulti stimolano le corrispondenti emozioni nel bambino il quale impara come siano piacevoli anche i comportamenti che rendono felici gli altri.

L'empatia non è però l'unico fattore per spiegare l'altruismo. Esistono infatti situazioni in cui l'empatia non entra in gioco e inoltre non necessariamente un comportamento altruistico è generato da una situazione empatica. L'empatia può in effetti stimolare la persona ad aiutare un suo simile in pericolo, ma nel momento in cui il primo decide se intervenire (ed essere quindi altruista) o meno (risultando egoista) entra in gioco una sorta di calcolo costi-benefici.

Lorenz sottolinea come i comportamenti animali siano il frutto di processi in cui intervengono stimoli diversi, spesso anche in contrasto tra loro. Dalle sue affermazioni si può evincere come le emozioni possano agire in direzioni diverse e come non sia corretto pensare che siano state selezionate allo scopo di garantire un comportamento altruistico.

Elster (1998), Hermailn e Isen (1999) sostengono come il rimorso conseguente ad un comportamento egoistico possa fungere da deterrente nei confronti di certe azioni: "reference to moods and other emotions in economics is rare. When such reference is made, it's usually to make sense of some behavior that seems inconsistent with narrow self interest. For example, honesty in situations where dishonesty would appear to have a larger payoff".

[15] L'empatia è una risposta emotiva provocata da uno stato d'animo o da una condizione vissuta da un altro individuo. Ad esempio, se osservo una persona che si trova in una situazione di pericolo, la reazione empatica mi farà provare uno stato d'animo simile al suo e mi potrà spingere ad aiutarlo.

Rabin (1993) inserisce in un contesto di teoria dei giochi[16] alcune nozioni psicologiche relative all'altruismo e al comportamento verso gli altri. Gli individui sono in genere generosi solo verso chi si è comportato allo stesso modo nei loro confronti ma tendono a comportarsi scorrettamente (anche a costo di subire delle perdite) con chi non è stato a sua volta equo[17]. Rabin dimostra inoltre che gli equilibri del modello impongono un comportamento simmetrico da parte degli individui. Sembra evidente, dunque, il legame con la strategia del *tit for tat*.

Tornando all'ambito dell'impresa, si può vedere come anche in questo particolare settore le emozioni giochino un ruolo fondamentale. Il desiderio di essere trattati onestamente è strettamente collegato all'impegno ed allo sforzo profuso dal lavoratore nel compiere il suo dovere, e se il lavoratore pensa di non essere trattato in modo corretto l'impresa non riuscirà ad ottenere la produttività desiderata. Si tratta insomma di una sorta di *do ut des,* di scambio di beni. Così anche secondo i modelli che analizzano il mercato del lavoro, un comportamento onesto può risultare più efficiente e di conseguenza premiare chi lo mette in atto (Novarese, 2000).

2.10 Il ruolo dell'altruismo nell'ambito economico

I numerosi lavori brevemente analizzati nei precedenti paragrafi hanno messo in luce il ruolo fondamentale che altruismo, cooperazione ed apprendimento hanno nei diversi settori di analisi, ma le osservazioni e le conclusioni raggiunte hanno anche un ruolo fondamentale per la comprensione dei differenti comportamenti degli agenti che interagiscono nel mondo economico.

Innanzitutto i diversi studi presi in esame hanno mostrato come i comportamenti altruistici siano il risultato di processi di una selezione che può avvenire sia a livello culturale che biologico inducendo a pensare ad una possibile interazione tra i due stadi. Inoltre è evidente come esistano differenti forme di altruismo determinate dalle diverse

[16] Anche Brams (1997) introduce le emozioni (in particolare la frustrazione) in una contesto di teoria dei giochi. In determinate situazioni, l'espressione, apparentemente irrazionale, della propria frustrazione può rivelarsi una scelta razionale in quanto consente di rendere credibile una minaccia (che se i soggetti agissero razionalmente non lo sarebbe).

[17] Il desiderio di punire chi si comporta disonestamente è evidenziato anche dagli esperimenti di Kahneman, Knetsch e Taler (1986, pp 105-108) e sembrerebbe emergere anche nelle prove di Andreoni (1995).

motivazioni che inducono a tale comportamento. In certe situazioni abbiamo rilevato un altruismo quasi automatico, in altre meno. Anche l'esperienza individuale e le molteplici sfaccettature delle diverse personalità danno origine a differenti valutazioni dell'equità.

Le esperienze empiriche tendono a mostrare un'inclinazione alla cooperazione, anche se non perfetta, ma che generalmente decresce al crescere del numero di interazioni tra gli individui. Ciò potrebbe indurre a pensare che l'altruismo evidente in partenza sia almeno parzialmente strumentale e dettato da motivi strategici. Ma anche la percezione che gli altri individui non stiano cooperando ed il conseguente desiderio di rivalsa potrebbe essere una valida interpretazione dei risultati ottenuti dagli esperimenti effettuati.

Un problema che abbiamo riscontrato anche noi nell'analisi e interpretazione dei risultati degli esperimenti effettuati a Torino è relativo alla misura in cui il "laboratorio" (nel nostro caso si tratterà di un'aula di informatica) riesce a limitare le eventuali influenze sociali e culturali alla base del comportamento altruistico. Il rischio che i risultati ottenuti siano distorti è elevato e va tenuto in considerazioni prima di trarre conclusioni affrettate.

Abbiamo poi rilevato come l'altruismo sia un elemento essenziale anche per la teoria dell'impresa. Proprio questo tipo di comportamento può essere un'efficace alternativa ai problemi di agenzia. Le diverse motivazioni dei lavoratori possono dunque essere un argomento fondamentale per la spiegazione e l'interpretazione delle diverse performance delle imprese e dei sistemi economici.

Con questo capitolo abbiamo quindi sottolineato l'importanza di approfondire la comprensione dei molteplici fattori alla base dei comportamenti non egoistici, frequentemente rilevati nei diversi ambiti analizzati e ricorrentemente individuati nei vari esperimenti svolti.

PARTE TERZA

Uno strumento per le simulazioni: Swarm

3.1 Il progetto Swarm

Il progetto Swarm è nato nel 1994 dagli studi di Chris Langton presso il Santa Fe Institute (New Mexico, USA). Anche Roger Burkhart, Nelson Minar, Glen Ropella (Swarm Corporation), Manor Askenazi, Irene Lee, Vladimir Jojic, e Alex Lancaster (Santa Fe Institute) hanno collaborato intensamente alla progettazione ed alla realizzazione di Swarm. L'attuale responsabile del progetto è Marcus Daniels che dirige anche lo Swarm Development Group (SDG), un'organizzazione *no profit* che si occupa di aggiornare e sviluppare continuamente il software. I ricercatori e i programmatori che da anni lavorano a tale programma hanno creato un insieme di librerie e di strumenti standard per lo sviluppo di modelli di simulazione multi-agente tali da consentire l'analisi di sistemi complessi di comportamento nell'ambito delle scienze naturali e sociali.

Swarm permette di modellizzare il comportamento di un insieme di "agenti" indipendenti al fine di ottenere un andamento collettivo che segue rigorose regole. Per questa sua caratteristica può essere utilizzato in moltissime applicazioni quali, per esempio, modelli economici, applicazioni di intelligenza artificiale, legami chimici tra molecole, ecc.

Gli elementi fondamentali che formano un'applicazione Swarm sono gli "agenti" e il "mondo". I primi vengono definiti dal programmatore e costituiscono la struttura del sistema che si vuole realizzare, mentre il mondo rappresenta lo spazio in due dimensioni dove gli agenti si muovono. Il comportamento di ogni singolo agente viene modificato a seconda dello stato del mondo istante per istante.

Il software è inoltre dotato di una precisa gestione dell'evento tempo per mezzo di una perfetta sincronizzazione delle azioni interna al programma e consente di interrogare in qualunque istante il sistema sullo stato delle diverse variabili per mezzo di sonde di non complessa programmazione.

Grazie alla piattaforma Swarm gli studiosi sono oggi in grado di concentrarsi sullo sviluppo delle applicazioni senza imbattersi nei tipici problemi della programmazione

informatica (come ad esempio la generazione di grafici o di un'interfaccia utente, il salvataggio e l'archiviazione dei risultati delle simulazioni).

In sintesi le tre principali caratteristiche di Swarm sono le seguenti:

- Il codice Swarm è orientato agli oggetti. Le librerie Swarm sono scritte nel linguaggio Objective C, un'evoluzione del C. L'Objective C permette di creare delle classi dalle quali possono essere generate delle istanze individuali. Queste istanze sono entità auto-contenute e la programmazione orientata agli oggetti risulta essere molto utile per la creazione di modelli basati sugli agenti.

- I programmi Swarm sono "gerarchici". La maggior parte delle applicazioni Swarm sono costituite da una struttura di questo tipo: il livello principale, l'"*Observer Swarm*", che genera il secondo livello, il "*Model Swarm*" e l'interfaccia utente. Il *Model Swarm* genera gli agenti individuali, scheda le loro attività, raccoglie informazioni sul loro operato e, se l'*Observer* lo richiede, trasmette tali dati al livello superiore.

- Swarm fornisce molti validi strumenti. Le librerie Swarm forniscono molte parti di codice che semplificano la creazione di un modello ad agenti. Questi strumenti facilitano la gestione della memoria, la conservazione dei dati nelle liste, la raccolta delle azioni e molti altri compiti.

3.1.1 Una piattaforma dinamica

Il software può essere scaricato direttamente dal sito www.swarm.org e viene rilasciato in licenza *open source* (sotto la GNU *General Public License*). Proprio grazie alla caratteristica di completa osservabilità del codice sorgente, Swarm è in continua evoluzione. Sono nati infatti numerosi gruppi di utilizzatori che contribuiscono costantemente all'ampliamento e al perfezionamento della piattaforma eliminando *bug*, aggiungendo parti di codice o introducendo nuove caratteristiche.

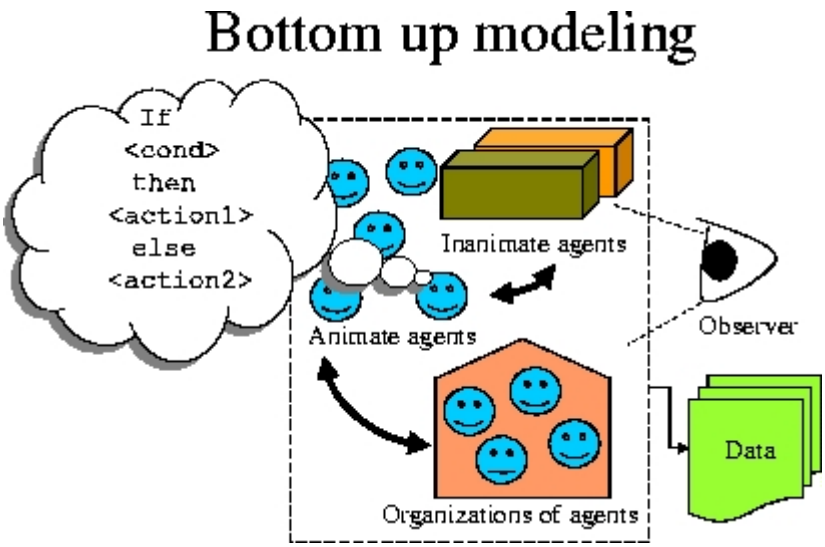
Il progetto è nato per sistemi operativi Unix in grado di supportare il sistema X Windows e la prima versione è stata rilasciata nel 1995. Nel gennaio del 1997 il software può essere utilizzato anche sui sistemi operativi Solaris e Linux e poco tempo dopo diventa disponibile una versione anche per la piattaforma DEC Alpha. Nell'aprile del 1998, con l'aiuto del package Cygnus Win32, anche sui sistemi Microsoft Windows 95/NT è possibile l'installazione del software di simulazione.

Come detto, le librerie Swarm sono scritte in Objective C, ma dal 1999, sempre grazie ai continui interventi degli utilizzatori della piattaforma, è stata creata una libreria che consente la programmazione in Java.

Swarm è quindi il frutto di una costante e continua interazione tra comunità di ricercatori e proprio per questa ragione non è possibile prevedere quale sarà lo sviluppo futuro del programma.

3.2 Swarm e la programmazione ad oggetti

Swarm è stato progettato per aiutare i ricercatori a costruire modelli nei quali attori semplici interagiscono dando origine a "sistemi complessi". Compito dei programmatori è quindi la creazione di "agenti", entità autonome, dotate eventualmente di capacità di decisione e di apprendimento. Particolari tecniche matematiche quali ad esempio le funzioni a reti neurali, gli algoritmi genetici, i *classifier system*, gli automi cellulari, ecc. permettono di simulare l'apprendimento dei vari agenti, siano essi investitori, api, formiche o gli onnipresenti "*bug*". Uno dei maggiori successi di Swarm è proprio la possibilità di ottenere modelli generali partendo dall'interazione di entità semplici costruite con attenzione ai singoli comportamenti individuali. Ogni agente è dotato di particolari caratteristiche comportamentali e l'interazione dei diversi agenti consente di dedurre il comportamento dell'intero sistema (*bottom up modeling*).



Proprio la programmazione ad oggetti consente di rappresentare modelli di questo tipo. Come si può notare dall'immagine gli oggetti sono "auto-contenuti" e sono stati elaborati in modo da renderli capaci di trasmettersi informazioni (rispondere a domande), che possono anche essere elaborate, suddivise in diverse categorie e archiviate.

.

3.2.1 Introduzione alla programmazione ad oggetti

L'*Object-Oriented Programming* (OOP) è il paradigma della programmazione attualmente dominante che ha sostituito le tecniche di programmazione "strutturate" e basate sulle procedure, sviluppate nei primi Anni 70. L'OOP, in termini semplici, è una tecnica di programmazione che si concentra sui dati (oggetti) e sulle relative interfacce. Per seguire un'analogia con la falegnameria, un falegname "orientato agli oggetti" si preoccuperebbe principalmente della sedia che sta costruendo e secondariamente degli strumenti impiegati per crearla, mentre un falegname "non orientato agli oggetti" penserebbe prima agli strumenti.

Un programma in Java o in C++ implementato dalla OOP è costituito da una serie di oggetti con determinate proprietà e caratterizzato da una serie di operazioni che tali oggetti sono abilitati a svolgere. Lo stato dell'intero sistema può cambiare nel tempo, ma il ricercatore può sempre contare sul fatto che gli oggetti interagiscono tra loro in modo documentato.

Ciò che contraddistingue in modo netto la programmazione strutturata tradizionale (che consiste nell'impostare una serie di funzioni, algoritmi, per risolvere un problema) dall'OOP è l'assegnazione a ciascun oggetto della responsabilità dell'esecuzione di una serie di attività correlate. Se un oggetto si basa su un'operazione che non rientra nelle sue responsabilità, deve avere accesso a un oggetto le cui responsabilità comprendono tale operazione. Il primo deve quindi chiedere al secondo di eseguire l'operazione a lui interdetta.

Un oggetto contiene due tipi di informazioni:

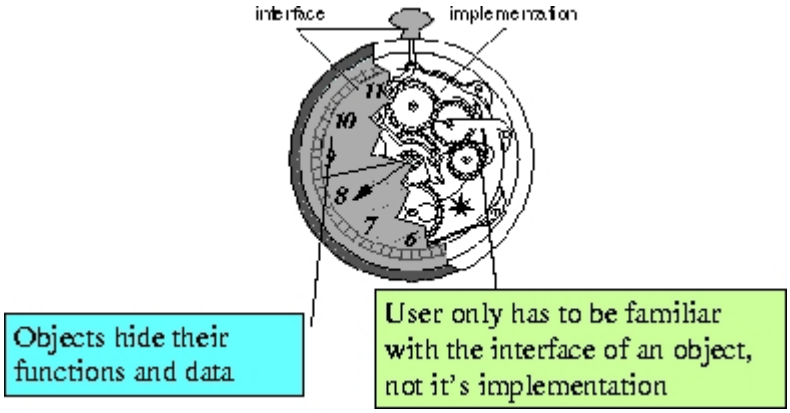
- *Variabili*. La lista di variabili riassume lo "stato" dell'agente, la sua età, lo stato di salute, le sue capacità, ecc. Le variabili possono essere di tipo intero (*int*), numeri a virgola mobile (*float),* vettori, puntatori, ecc.

- *Metodi*. I metodi determinano cosa un oggetto è effettivamente in grado di fare. Tipicamente potrebbero esserci metodi che ricevono informazioni dal mondo esterno, metodi che spediscono informazioni all'esterno e metodi che elaborano informazioni.

La programmazione ad oggetti è caratterizzata da quattro proprietà essenziali:

- *Astrazione.* Il codice che descrive un oggetto è scritto in una classe e nel programma principale gli oggetti sono creati come esempi, istanze, della classe.

- *Incapsulamento.* Gli oggetti nascondono i propri metodi e dati, separando l'interfaccia dell'implementazione. In pratica si può vedere cosa fa un oggetto, ma le regole che lo guidano sono definite da un'altra parte. L'oggetto può essere così modificato senza dover stravolgere le parti di codice che lo utilizzano.



- *Ereditarietà.* Ogni classe di oggetti può essere derivata da altre classi già definite. In questi casi, la classe "figlia" eredita tutte le variabili e i metodi della classe "madre", li può modificare e ne può aggiungere altri.

- *Polimorfismo*. Si possono avere metodi con lo stesso nome, ma che hanno parametri diversi (*overloading*).

Si deve notare che Swarm non permette il polimorfismo. I metodi creati in Swarm devono dunque avere nomi unici.

3.3 Le librerie Swarm

In aggiunta alle librerie che Swarm utilizza per la compilazione bisogna considerarne altre che permettono la creazione dei modelli bidimensionali, di algoritmi genetici e di reti neurali.

Di seguito sono brevemente descritte alcune caratteristiche dei package tipici di Swarm.

All'interno di ogni libreria sono implementati gli oggetti che grazie ai loro metodi permettono la caratterizzazione di ogni agente che si crea nel modello. Ognuno di questi metodi svolge delle funzioni particolari che possono essere richiamate dal programmatore includendo la libreria nel suo codice sorgente e utilizzando il metodo riferito all'agente.

3.3.1 Defobj Library

La *Defobj Library* sostiene l'intera impalcatura della programmazione orientata agli oggetti tipica di Swarm. Definisce un preciso stile da utilizzare nella programmazione con Objective C che include anche le convenzioni standard necessarie per la creazione degli oggetti, per l'allocazione delle variabili in memoria, per la gestione dei messaggi d'errore, per l'utilizzo delle funzioni di *debug* (ricerca degli errori).

3.3.2 Collection Library

Gli oggetti appartenenti alla *Collection Library* hanno il compito di permettere la catalogazione dei riferimenti agli oggetti e di altri valori in genere all'interno di particolari strutture (*Array, List, Map, Set*). Tali strutture sono definite dalle loro interfacce e non dall'implementazione interna. Esistono tuttavia particolari opzioni che permettono una maggiore flessibilità ed un controllo più efficiente, come ad esempio la possibilità di implementare altre librerie.

.

3.3.3 Activity Library

L'*Activity Library* è responsabile del susseguirsi delle azioni all'interno del mondo simulato e consente che tali azioni si verifichino sempre al momento giusto e nell'ordine stabilito. Costituisce il fondamento della dinamica della simulazione orientata agli oggetti tipica di Swarm. Le azioni sono costituite da messaggi agli oggetti, chiamate alle funzioni o gruppi di azioni in un preciso ordine predefinito.

L'*Activity Library* garantisce che l'insieme di queste azioni, e di tutte le variazioni del sistema che esse producono, avvengano in un preciso e prevedibile ordine temporale. Il tempo è definito dall'ordine relativo delle azioni e può anche essere indicizzato per mezzo di valori discreti generati da un elemento orologio.

3.3.4 Objectbase Library

L'*Objectbase Library* include molti aspetti fondamentali del modello ad oggetti di Swarm e definisce "l'impianto esplorativo" (*probing machinery*) utilizzato per raccogliere i dati dai vari oggetti.

Le sonde (*probe*) sono delle entità che consentono all'utente della simulazione di monitorare e modificare la struttura interna degli oggetti durante lo svolgimento della simulazione stessa, senza dover quindi operare in fase pre-compilativa. Le sonde consentono un'interazione dinamica tra l'istanza di un oggetto, le variabili e i metodi. La maggior parte delle funzioni relative alle sonde è implementata in questa libreria, che è però strettamente collegata con la *Simtools Library* che consente la visualizzazione delle sonde per mezzo dell'interfaccia grafica.

3.3.5 Random Library

In questa libreria sono implementati alcuni algoritmi per la generazione di numeri random con distribuzioni di tipo uniforme o gaussiana. Grazie ai suoi metodi è possibile settare il *range* di valori e il tipo di algoritmo: SWB (*Subtract with Borrow lagged* Fibonacci), LCG (*Linear Congruential Generator*), ACG (*Additive Congruential Generator*) e altri.

3.3.6 Simtools Library

Simtool è la libreria Swarm per antonomasia. E' la libreria nella quale sono state inserite classi di vario genere che, nonostante siano di fondamentale importanza, non sono sufficientemente specifiche da essere inserite nelle altre librerie. Le classi contenute in *Simtool*, per esempio, possono essere utilizzate per creare un'interfaccia con il mondo esterno, per consentire il caricamento di dati da un file, per il salvataggio di dati relativi alle azioni compiute dal singolo agente e per avere a disposizione ulteriori supporti di grafica.

3.3.7 Gui Library

Tcl/Tk è la combinazione di un linguaggio di programmazione (*Tcl, Tool Command Language*) e di una libreria grafica (*Tk, Tool Kit*) per Linux. Tale strumento permette di creare una semplice interfaccia grafica nascondendo all'utente le molto più specifiche parti di codice *Tk*. Per creare e utilizzare i *widget*[18] l'utente deve semplicemente utilizzare degli oggetti. Alcuni di essi sono strettamente legati ai *widget Tk*, altri sono la combinazione di diversi *widget* oppure vengono creati con nuove parti di codice. La *Gui Library* è una fondamentale libreria che consente all'utente di creare un'avanzata interfaccia utente senza la necessità di conoscere ulteriori linguaggi di programmazione al di fuori dell'Objective C o di Java.

3.3.8 Analysis Library

Questa è una libreria che permette la creazione di grafici relativi ad eventuali parametri della simulazione con la possibilità di effettuare operazioni statistiche su serie di dati quali media, entropia, determinazione del massimo o del minimo.

3.3.9 Space Library

La *Space Library* è uno strumento utile per assistere l'utente nella creazione dell'ambiente in cui dovranno interagire gli agenti. In generale lo spazio può essere variabile quanto gli agenti stessi (l'ambiente può essere infatti visto come un ulteriore agente). Nonostante l'ampio spettro di ambienti che si possono avere, molte simulazioni hanno caratteristiche simili che possono essere raggruppate in un codice generico. L'attuale *Space Library* è dotata di attributi spaziali semplici di tipo bidimensionale discreto. Se l'utente ha bisogno di incrementare o modificare tali proprietà può farlo aggiungendo parti di codice a seconda della necessità. Alla classe *Space Library*

[18] Elemento dell'interfaccia utente. Un tempo slang dei programmatori, ora termine tecnico di tutto rispetto. Pulsanti, barre di scorrimento, campi di modifica, elenchi ad albero e così via sono tutti *widget*. I programmatori combinano questi elementi nei programmi che funzionano su desktop grafici come GNOME e KDE. I *widget* sono definiti nei *toolkit* come *tk* (il primo e originale), *Qt* e *Gtk.*

possono essere applicati molti metodi tra i quali per esempio il *Discete2d*, che consente di definire con precisione le coordinate bidimensionali dell'ipotetico punto in cui si trova l'agente all'inizio della simulazione e di conoscere in un qualsiasi momento successivo la corretta nuova posizione dell'agente virtuale.

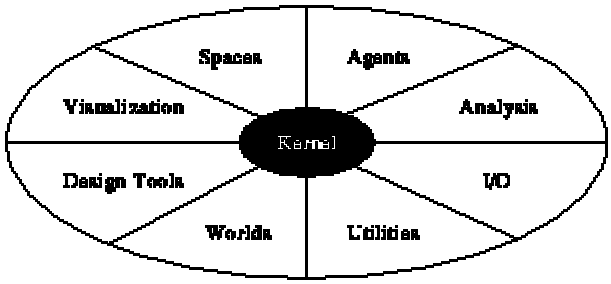
3.3.10 Probe Library

Fanno parte di questa libreria il *Probes*, *ProbeMaps* e *ProbeDisplay* che servono per le interazioni dinamiche tra gli oggetti in una simulazione. Tali librerie permettono di impostare e leggere eventuali variabili o parametri che possono cambiare istante per istante. Inoltre è possibile visualizzare tali valori in caselle grafiche in *real time*; per esempio, il valore della costante di diffusione, il numero in oggetti creati, il tempo trascorso ecc.

3.4 La struttura di una simulazione

La creazione di una qualsiasi applicazione Swarm richiede la conoscenza del linguaggio di programmazione Objective-C o di Java.

Swarm è caratterizzato da un insieme di librerie orientate agli oggetti raggruppate tutte in un unico *kernel*.



Creare un'applicazione significa costruire una collezione di agenti che seguono determinati criteri o eventi che devono far parte di uno *schedule*, ovvero di una struttura formata da una sequenza di azioni da eseguire. Le azioni svolte da un singolo o da un

gruppo di agenti sono selezionate in base all'informazione contenuta nel messaggio che il *kernel* spedisce dallo stato della simulazione istante per istante formando in questo modo una rete di connessioni che permette il cambiamento di stato di un agente.

Ogni livello del programma è volutamente indipendente dagli altri e i legami diretti tra essi sono evitati al fine di consentire una più efficace manipolazione del listato sia nella fase preliminare, durante la prima elaborazione della simulazione, sia in epoche successive, per modificare parti di codice .

I ricercatori commentano i passaggi chiave della struttura di un listato per permettere una più rapida e precisa rilettura, importante sia per consentire al programmatore stesso una maggior facilità nella rilevazione di eventuali errori, sia per permettere a terzi di avere un più semplice accesso alle diverse parti del programma.

Si tratta quindi di una forma di standard di progettazione indispensabile per consentire allo Swarm di evolversi. Rispettando tali caratteristiche di programmazione è possibile mettere a disposizione della comunità di Swarm ogni simulazione. Ognuno è così in grado di contribuire attivamente all'evoluzione del progetto di Santa Fe.

3.4.1 Model Swarm

Il *Model Swarm* permette di contenere la vera implementazione in codice del modello che si desidera simulare. E' formato da un set di segnali di ingresso e di uscita. I primi rappresentano i parametri (numero di agenti, coordinate bidimensionali degli agenti nel mondo, parametri di diffusione, le azioni degli agenti, lo *scheduler* delle azioni, eventuali strutture correlate con il modello), mentre le uscite rappresentano i valori dei parametri a seconda dello stato. La parte fondamentale del movimento di un singolo agente è rappresentata all'interno dell'*ActionGroup* che permette di valutare i segnali di ingresso e di selezionare il movimento relativo.

3.4.2 Observer Swarm

Tale oggetto permette di osservare e misurare una simulazione, ovvero calcola i parametri di ingresso e di uscita definiti nel *Model*. Le attività dello *schedule*

dell'*Observer* consistono nel ricevere come segnali di ingresso le uscite del *Model* e creare opportuni grafici. Per questa sua attività all'interno dell'*Observer* è necessario richiamare delle librerie di grafica che permettono la creazione del mondo e dei singoli oggetti. Tali grafici possono essere creati al primo *step* dall'istanza *EZGraph* e dalle sue azioni, mentre ai passi successivi viene utilizzato AverageSequence che disegna il grafico in base al valore di uscita e consente l'aggiornamento istante per istante.

3.4.3 Main.m

Il *Main* o *Start* rappresenta l'oggetto di partenza dove si crea la sequenza di istanze da avviare. Crea il pannello per la gestione della simulazione e lo rende attivo.

3.4.4 RuleMaster

Il *RuleMaster* è l'oggetto che contiene tutte le regole di comportamento che dovranno seguire gli agenti ed è stato inserito nella piattaforma dal Gruppo Torinese Utilizzatori di Swarm (prof. P. Terna).

Viene organizzato come un magazzino di regole. Ogni situazione che si vuole gestire deve essere formalizzata e destinata a portare a termine un compito preciso. Ogni agente, definito in altri file, interroga il *RuleMaster* per sapere quali scelte compiere e solo dopo aver ottenuto una risposta agisce.

3.4.5 Batch Swarm

Questo oggetto sostituisce l'*Observer* Swarm nel caso in cui non si disponga di un'interfaccia grafica e permette di registrare i risultati della simulazione in appositi file.

3.5 Alcuni oggetti importanti: liste, mappe e vettori

Liste (*Lists*), mappe (*Maps*) e vettori (*Arrays*) sono una collezione di oggetti di Swarm che consentono al ricercatore di tenere una traccia di tutte le azioni che gli agenti, artificiali o umani che siano, compiono durante lo svolgimento della simulazione. Grazie a questi strumenti sarà possibile in un qualunque momento successivo alla simulazione verificare come un dato agente si è comportato in una determinata situazione.

3.5.1 Le liste

Il protocollo *List* di Swarm implementa una lista con dimensioni flessibili (cioè le dimensioni della lista crescono quando sono aggiunti elementi, e decrescono quando il numero degli elementi è inferiore). La lista permette di elencare oggetti, e di eseguire, in modo semplice, azioni ripetitive sugli oggetti contenuti nella lista.
Una lista si crea in Java con la seguente stringa:

nameOfList= new ListImpl(getZone())

Alla lista possono essere applicati diversi metodi:

- *addFirst, addLast* aggiunge un oggetto all'inizio o alla fine della lista.

- *removeFirst, removeLast* cancella il primo o l'ultimo elemento della lista.

.

- *getCount* riporta il numero di elementi nella lista.

- *removeAll* cancella tutti gli oggetti della lista, ma non la lista stessa.

- *deleteAll* cancella tutti gli oggetti della lista, e cancella anche la lista dalla memoria.

- *atOffset(i)* per conoscere l'elemento che si trova in posizione `i` nella lista.

IMPORTANTE: nelle collezioni di Swarm, la numerazione degli elementi inizia dallo zero. Dunque, il primo elemento della lista si trova alla posizione 0, il secondo elemento si trova alla posizione 1, ecc.

3.5.2 I vettori

I vettori vengono utilizzati quando si ha bisogno che gli oggetti siano catalogati in un ordine specifico. La maggior differenza che si può riscontrare rispetto alla lista è che il vettore viene dichiarato con la dimensione. Non è possibile aggiungere oggetti non compresi in questa dimensione (quindi non si possono usare i metodi *addFirst* o *addLast*). Poiché anche la dimensione del vettore non può essere modificata non è consentito cancellare elementi dal vettore.

Il vantaggio che si riscontra nell'utilizzo del vettore rispetto ad una lista è che gli oggetti possono essere trovati molto velocemente in quanto il loro indice è un numero intero.

Il vettore viene generato con la seguente riga di codice:

nameOfArray= new ArrayImpl(getZone,size);

Ai vettori possono essere applicati i seguenti metodi:

- *.atOffset$put(i,obj)* per mettere l'oggetto obj alla posizione *i* del vettore.

IMPORTANTE: anche nel caso dei vettori la numerazione incomincia dallo zero.

- *.atOffset(i)* per avere l'oggetto che si trova alla posizione *i* del vettore.

- *GetCount*
- *.contains(aMember)*
- *.forEach(selector)*

3.5.3 Le mappe

La mappa può essere vista come una tabella con due file di elementi. Nella prima si trovano i nomi degli oggetti mentre nella seconda fila si trovano gli oggetti che vogliamo conservare nella collezione. Quindi, per inserire un oggetto in una mappa è necessario specificarne il nome (*key*, cioè chiave).

E' possibile poi chiedere alla mappa di restituire o cancellare l'oggetto che corrisponde ad una chiave particolare.

E' importante ricordare che in Swarm anche le chiavi devono essere oggetti.
Una mappa si crea nel modo seguente:

nameOfMap= new MapImpl(getZone())

Alle mappe possono essere applicati i seguenti metodi:

- *.at$insert(keyObject,object*) per inserire *object* con la chiave *keyObject.*
- *at(keyObject*) per ottenere l'oggetto corrispondente a *keyObject.*
- *GetCount*
- *.contains(object)*
- *.forEachKey(selector)*

- *getFirst,getLast*

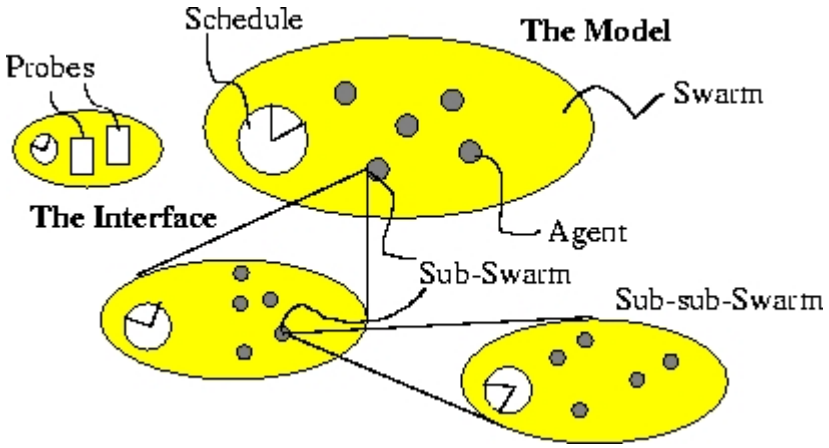
3.6 Schedules e Actions

Schedule e *action* consentono di integrare le diverse azioni dei vari agenti, siano essi umani o artificiali nei diversi livelli della simulazione. Sono quindi alla base di una delle caratteristiche più importanti e di maggior interesse di Swarm.

3.6.1 Schedules

Uno *schedule* è come un ordine del giorno nel quale si scrive cosa fare ad ogni ora. Uno *schedule* è dunque un altro oggetto di Swarm nel quale possiamo inserire messaggi individuali, o gruppi di messaggi (*ActionGroups*). La gestione della durata dell'intervallo fra le azioni è fondamentale; generalmente, il *repeatInterval* vale 1, il che vuol dire che le azioni sono ripetute ogni periodo. In Java il valore del *repeatInterval* è il secondo argomento del costruttore dello *schedule*.

Una volta creato lo *schedule* (generalmente nel metodo *buildActions*) tutti gli oggetti della simulazione sono pronti e devono essere solo attivati.



3.6.2 La classe Activity

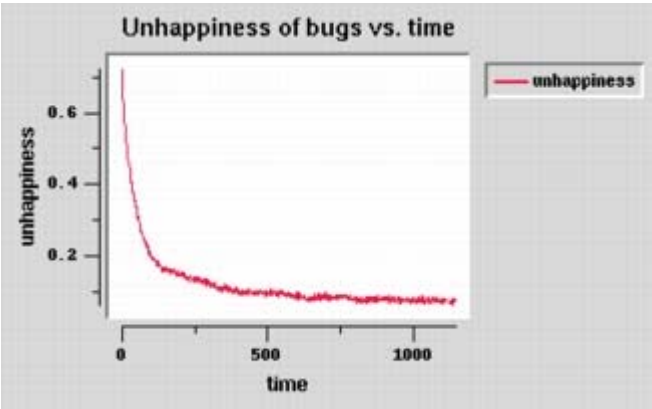
Tramite i metodi *activateIn* le librerie di Swarm integrano le azioni degli oggetti della simulazione. Il metodo *main()* segnala allo Swarm più alto nella gerarchia (nel nostro programma, il *Model Swarm*) di attivarsi in *null* (Java), dopo aver creato le sue azioni e i suoi *schedule*. Questo fa sì che il *Model Swarm* segnali al prossimo livello di Swarm di attivarsi, ecc; in questo modo, vengono sincronizzate le attività ai vari livelli della simulazione.

I metodi *activateIn* richiamano oggetti della classe *Activity*. Sul piano pratico tale classe risponde a metodi come *run, stop, next*, terminate, ecc, ma generalmente non viene usata direttamente; è il *ControlPanel* che permette di gestire la simulazione dal mouse e che nasconde gli oggetti della classe *Activity.*

3.7 L'interfaccia grafica di Swarm

Swarm fornisce un certo numero di classi e protocolli in grado di generare l'interfaccia grafica utente (GUI, *Graphic User Interface*) che include:

Grafici a linea (in questo caso una serie di tempo)



Istogrammi

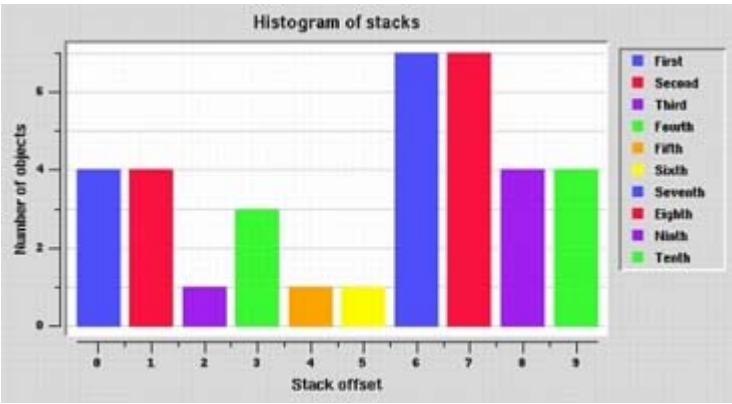
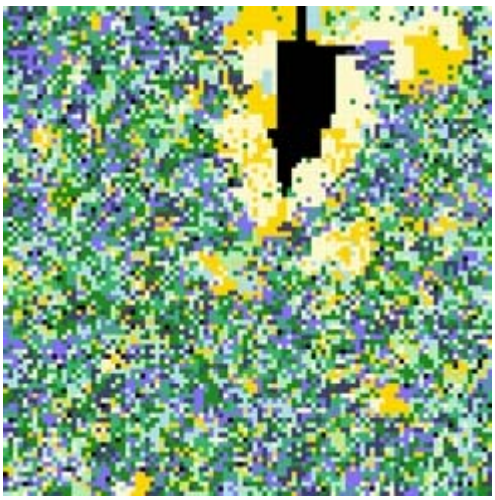
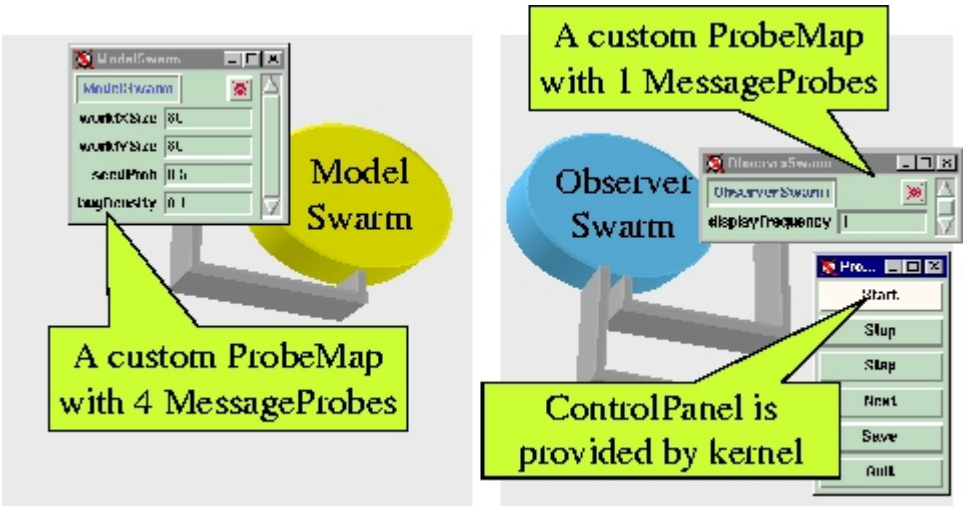


Grafico con dati discreti bidimensionali



Esempi di ProbeMaps per il Model Swarm e l'Observer Swarm



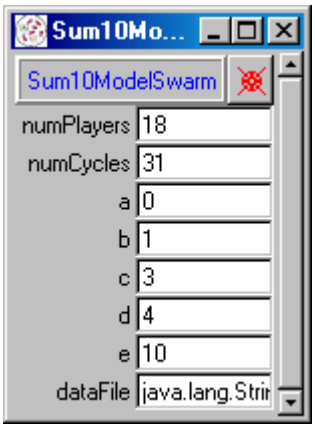
L'*Observer Swarm* è una classe che consente al ricercatore di osservare e controllare l'andamento della simulazione.

L'immagine mostra il *ControlPanel,* caratteristica principale dell'*Observer Swarm,* strumento che consente di interagire con la simulazione. Tale pannello è dotato di cinque bottoni:

- *start*: serve a dare inizio alla simulazione (si ferma solo se è prevista una fine)

- *stop*: mette in pausa la simulazione
- *next*: la simulazione compie un passo singolo (*step*)
- *save*: salva la disposizione delle finestre sullo schermo
- *quit*: termina la simulazione (anche se non è ancora stato raggiunto il numero di cicli previsto).

CustomProbeMap



Grazie all'*Observer Swarm* è possibile creare delle sonde (*Probes*) per visualizzare i valori dei parametri degli oggetti della simulazione. E' possibile inoltre cambiare tali valori prima di eseguire la simulazione (è infatti sufficiente modificare il valore nella finestra e premere il tasto '*return*').

Risulta pertanto immediatamente evidente un ulteriore vantaggio della programmazione ad oggetti: è altrettanto semplice operare con tre, dieci o cento agenti.

3.8 Alcuni messaggi d'errore

Durante la compilazione dei listati scritti dal programmatore è frequente imbattersi in messaggi d'errore.

Tali messaggi sono principalmente di 2 tipi:

- *_syntax*: errori di sintassi

- *semantic*: errori nell'uso di funzioni di Java o JavaSwarm, a volte collegati ad errori di sintassi.

I messaggi di errore in Java sono generalmente chiari, vengono infatti proposte dal compilatore stesso delle soluzioni. Inoltre Java fornisce indicazioni circa il file e la riga in cui l'eventuale errore è stato riscontrato.

Nella maggior parte dei casi la correzione dei primi errori implica una drastica riduzione dei messaggi del compilatore in quanto una imprecisione nelle prime righe di codice può implicare una lunga serie di errori ad essa correlati.

E' inoltre possibile che alcuni errori non vengano individuati in fase di compilazione ma solo più tardi durante l'esecuzione del programma. L'inserimento di blocchi *try/catch* consente però di evitare alcuni errori di *runtime*.
Alcuni errori che compaiono più frequentemente sono:

- *Error: No match was found for constructor "constructorName(list of arguments)"*

- *Error: No match was found for method "methodName(list of arguments)"*

E' necessario controllare che gli argomenti siano del tipo voluto, nell'ordine voluto, e che siano stati inseriti tutti (se il costruttore o il metodo è stato dichiarato per avere quattro argomenti, es. *initSwarm*(), comparirà questo messaggio di errore se ne sono stati utilizzati meno di quattro o di più). Bisogna però notare che alcuni costruttori o metodi possono essere dichiarati con diversi numeri di argomenti (*overloading*).

- *SyntaxError: "ClassName.methodName;" : "AssignmentOperator AssignmentExpression" inserted to complete statement expression*

methodName non ha argomenti, ma ha bisogno di parentesi vuote. Tali parentesi indicano a Java che *methodName* è una funzione.

- *SyntaxError: ”public type methodName”: ”;” inserted to complete FieldDeclaration*

Si tratta di un errore simile al precedente. Anche nella dichiarazione di un metodo senza argomenti è necessario includere le parentesi vuote.

- *Error: the method ”java.lang.Object methodName()” must contain a return statement compatible with type Object*

methodName() non è stato dichiarato come *void* e Java non ha trovato uno statement di return. Una soluzione è cambiare la dichiarazione da un tipo *Object* a un tipo *void*, oppure è necessario aggiungere *return this*.

PARTE QUARTA

Swarm Web Interface for Experimental Economics

4.1 Swiee: un'introduzione

Swarm Web Interface for Experimental Economics è il nome del software ideato e realizzato dal dottor Boero per dare la possibilità a ricercatori che operano nel campo delle scienze economiche e sociali di avere a disposizione un programma in grado di analizzare il comportamento di esseri umani che interagiscono con un mondo artificiale per analizzarne il comportamento.

L'utilizzo di Swarm nella creazione di modelli basati su agenti (*Agent Based Models*, ABM) dà la possibilità di costruire modelli in grado di rappresentare numerose situazioni reali, di riutilizzare il codice precedentemente scritto e consente ad altri ricercatori di sfruttare il proprio codice per riprodurre e replicare gli esperimenti.

L'economia sperimentale richiede l'utilizzo di programmi per permettere ai partecipanti all'esperimento di interagire nel mondo artificialmente creato, per generare tale mondo, per raccogliere i dati delle simulazioni e, infine, per consentire un'analisi ottimale dei risultati. Purtroppo, però, molti dei software al momento esistenti possono essere utilizzati in modo efficiente solo per certe categorie di esperimenti ed è per esempio molto difficile impiegare lo stesso programma per un esperimento sui beni pubblici e per un altro sull'organizzazione aziendale. L'obiettivo di Swiee, con l'aiuto di Swarm, è di evitare tali inconvenienti.

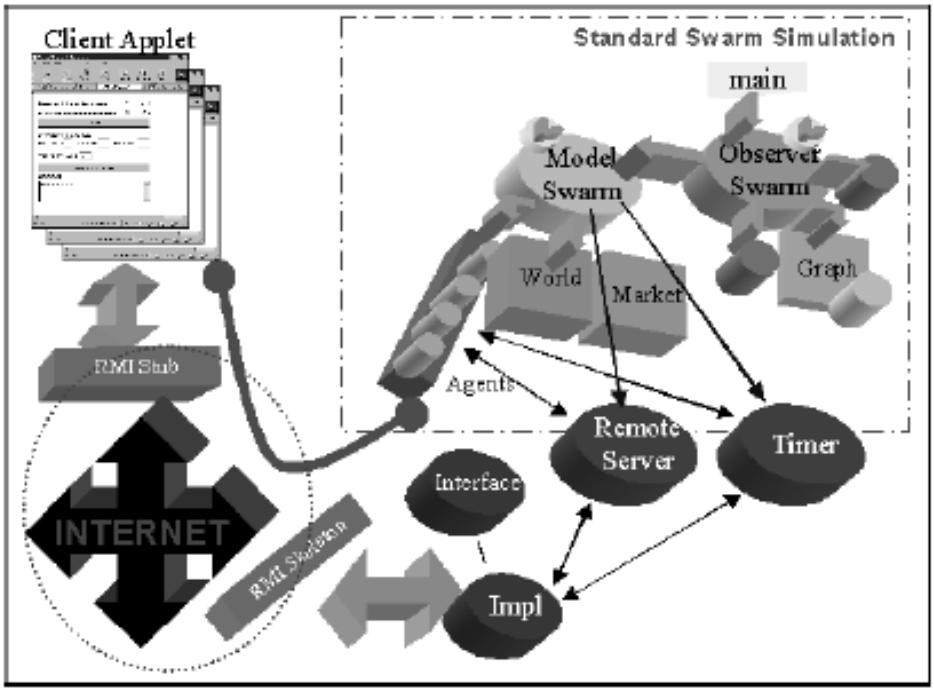
Una simulazione Swarm prevede generalmente un mondo artificiale che rappresenta un preciso mondo reale nel quale agenti artificiali, che rappresentano individui reali, interagiscono tra di loro. Il software realizzato ha anche lo scopo di consentire una diretta interazioni tra esseri umani e mondo virtuale, magari con l'intervento di agenti artificiali siano essi creati con semplici algoritmi, piuttosto che con l'impiego dell'intelligenza artificiale (reti neurali artificiali, algoritmi genetici, *classifier system*, automi cellulari, ecc.).

Java, possedendo un'architettura rigorosamente orientata agli oggetti ed essendo probabilmente il linguaggio di programmazione rivolto alle applicazioni di rete più semplice e completo, è stato ideale per la realizzazione di Swiee.

4.2 Alcune caratteristiche di Swiee

Swiee mostra chiaramente come sia semplice l'utilizzo di Swarm per la realizzazione di esperimenti economici o di carattere sociale. In particolare il software è dotato delle seguenti caratteristiche:

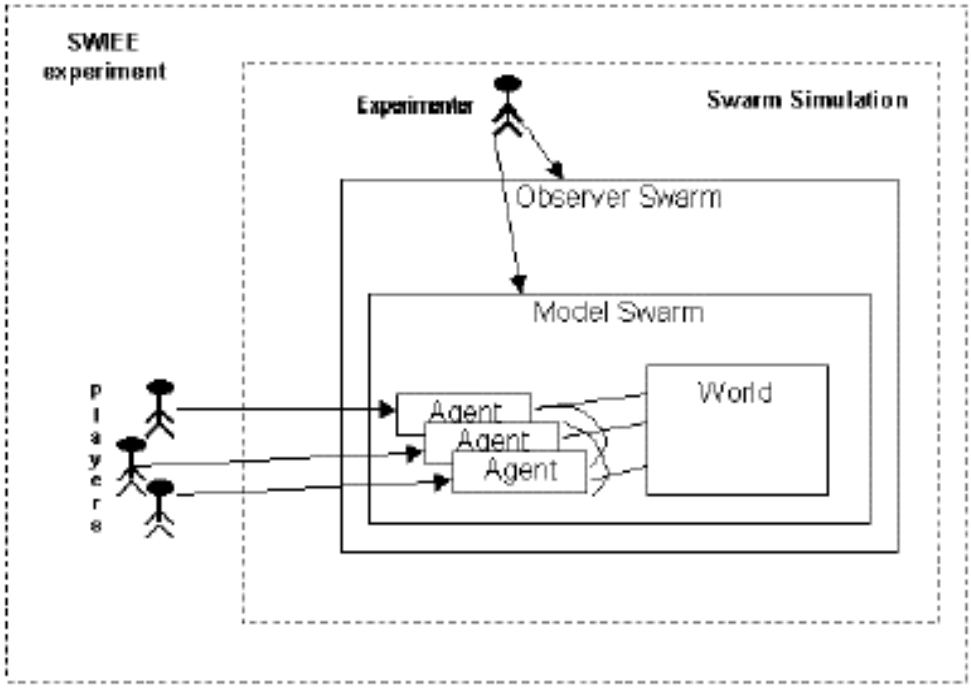
- è un progetto *open source*;
- alla sua base c'è Swarm, un programma di simulazione ormai molto diffuso;
- utilizza il linguaggio di programmazione Java;
- sfrutta i protocolli TCP/IP per le comunicazioni *client-server*;
- consente la realizzazione di qualunque tipo di esperimento con la possibilità, se necessario, di introdurre anche agenti artificiali;
- può essere impiegato sia per esperimenti di "laboratorio" che per esperimenti sull'internet.



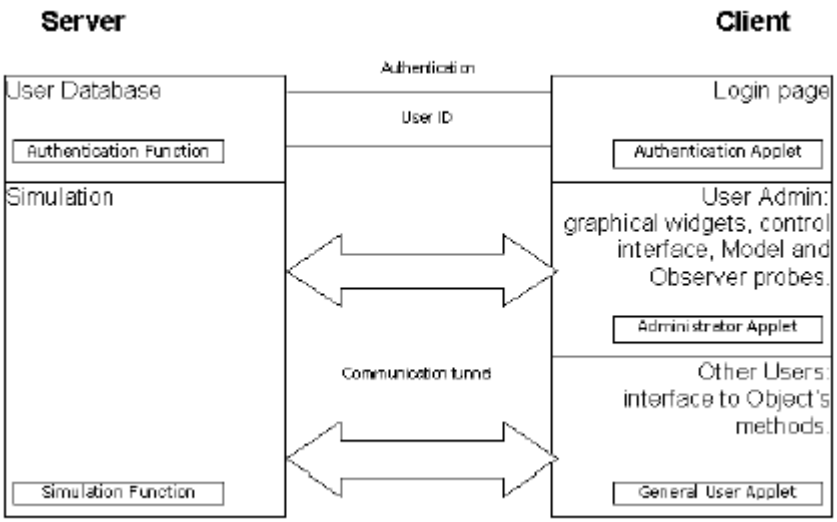
4.2.1 La struttura di Swiee

Il progetto Swiee segue fondamentalmente due diverse linee guida: da un lato la costruzione di esperimenti semplici e dall'altro lo sviluppo di un insieme completo di strumenti per gli esperimenti da svolgere con l'aiuto delle simulazioni. Per raggiungere tali scopi è indispensabile seguire due differenti approcci.

- Se si desidera mantenere la struttura standard semplice di Swarm, è sufficiente creare il collegamento tra il modello ad agenti e i giocatori umani.



- Il secondo approccio, invece, è sviluppato con strumenti più complessi che consentano allo sperimentatore di gestire ed automatizzare ogni operazione. Per operare in questa direzione è indispensabile gestire l'interazione tra le simulazioni Swarm e il *web server* utilizzando *server* e *database* di Java.



In questo secondo caso la struttura è basata su due differenti livelli: uno per l'autenticazione e l'altro per esecuzione, gestione e accesso all'esperimento.

4.3 Chiamate a metodi remoti (RMI)

Se l'obiettivo del progetto Swiee è di permettere l'interazione tra giocatori umani e il mondo virtuale di Swarm, è necessario riuscire a creare un collegamento remoto tra due applicazioni, il *client* da una parte ed il *server* con la simulazione Swarm dall'altra. Tale collegamento può essere costituito dall'internet, nonostante ci siano alcuni problemi relativi all'identificazione dell'utente, o dall'utilizzo di una rete locale (LAN).

Java risulta ancora una volta essere il linguaggio di programmazione ideale in quanto possiede numerose funzioni per la gestione dei protocolli di comunicazione TCP/IP. In Java è inoltre possibile la creazione di *applet*, programmi che operano sulla *Java virtual machine* (JVM) presente sul browser del *client*, non è così necessario scaricare programmi "ingombranti" sul *client* col rischio di appesantire il sistema. Bisogna infatti ricordare che i programmi Java non vengono compilati in linguaggio macchina, ma in un *byte-code* eseguibile dalla JVM. Tramite le chiamate a metodi remoti (RMI, *Remote Method Invocation*), particolari classi Java, è così possibile stabilire una comunicazione tra oggetti che operano su differenti JVM.

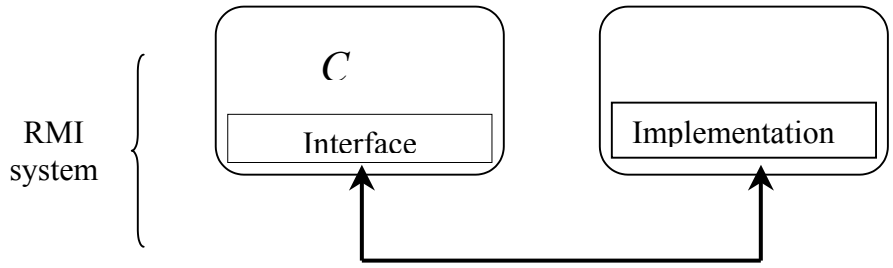
Il meccanismo RMI permette di gestire l'accesso ad un oggetto che si trova su una macchina diversa (es. sul *server*) e consente di richiamarne i metodi. I parametri del metodo devono essere comunicati in qualche modo all'altra macchina; all'oggetto deve essere chiesto di eseguire il metodo e il valore restituito deve essere inviato al *client*.

Quando si desidera richiamare un metodo remoto che si trova in un oggetto remoto, il codice *client* chiama un metodo ordinario di Java che è incapsulato in un oggetto surrogato, detto anche *stub*. Lo *stub* si trova sulla macchina *client*, non sul *server* e impacchetta i parametri usati dal metodo remoto in un blocco di *byte*. Questa operazione di formattazione dei parametri utilizza, per ciascun parametro, una codifica indipendente dal dispositivo. Gli oggetti sono codificati col meccanismo di serializzazione. Il processo di decodifica dei parametri viene chiamato ordinamento dei

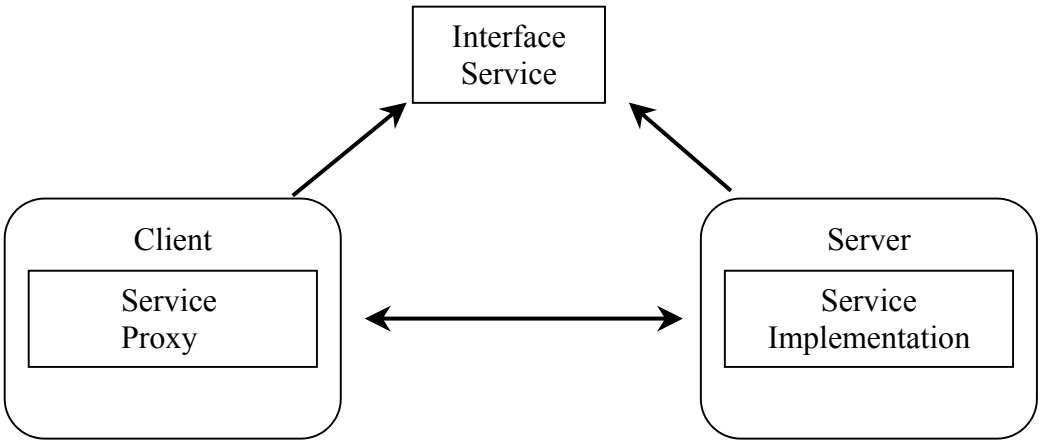
parametri e consente la conversione dei parametri stessi in un formato adatto al trasporto da una macchina virtuale all'altra.

Questo processo è molto complicato, ma avviene in maniera totalmente automatica e trasparente.

La caratteristica principale dell'architettura RMI consiste nel fatto che interfaccia e implementazione siano separate. Il *client* accede solo all'interfaccia mentre l'implementazione è contenuta nel *server*. L'interfaccia rappresenta la "definizione" di un servizio remoto, mentre l'implementazione del servizio è codificata in classi remote.



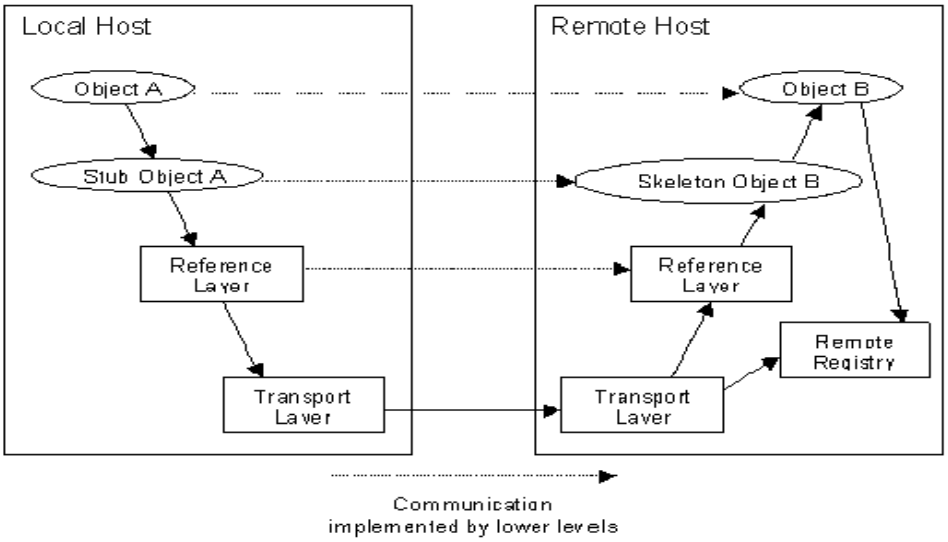
Per consentire agli oggetti del *client* di conoscere le interfacce del servizio remoto è necessario che sul *client* sia presente un'interfaccia *proxy*.



Un sistema delle RMI operativo è composto da molte parti diverse tra loro:

- La definizione dell'interfaccia per i servizi remoti

- L'implementazione dei servizi remoti
- I file *stub* e *skeleton*
- Un *server* che accolga i servizi remoti
- Un servizio RMI che permetta ai *client* di trovare i servizi remoti
- Un programma *client* che utilizzi i servizi remoti
- Un http *server* (un *web server*), se non si vogliono installare programmi sulle macchine *client*



4.4 Manuale di installazione di Swiee

L'installazione descritta nel seguente manuale è stata effettuata con le seguenti versioni dei programmi:

- Apache v.1.3 http://httpd.apache.org/dist/binaries/win32
- Swarm v.2.1.1 http://www.swarm.org/release-swarm.html

- JDK v.1.3.0 http://java.sun.com/j2se/1.3/download-windows.html

4.4.1 Apache

Apache è un progetto nato per creare un *webserver* stabile, affidabile e veloce per piattaforme Unix. Il *webserver* fa in modo che il PC diventi un *server web* (solo per quanto riguarda il protocollo HTTP, ovviamente; per altri servizi quali SMTP, FTP, POP e simili sarà necessario rivolgersi ad altri programmi); il *webserver* non è solo utile, ma addirittura necessario per l'esecuzione locale degli *script*: senza di esso, potremo solamente eseguire gli *script* direttamente dal *prompt* dei comandi, e non attraverso le pagine web come è necessario in molte applicazioni.

Configurazione di Apache:

Nella directory *c:\Programmi\ApacheGroup\Apache\Conf* aprire il file *Httpd.conf* e controllare la voce "MaxKeepAliveRequests" nel contesto:

```
# MaxKeepAliveRequests: The maximum number of
requests to allow
# during a persistent connection. Set to 0 to allow
an unlimited amount.
# We recommend you leave this number high, for
maximum performance.
MaxKeepAliveRequests 100
```

"100" indica il numero massimo di utenti connessi contemporaneamente, è importante verificare che sia sufficientemente elevato ed è possibile sostituire il numero con quello necessario.

Nello stesso file sostituire alle voci "*Document Root*" la *root* di Swarm.
Per esempio, così:

```
# DocumentRoot: The directory out of which you will
serve your
# documents. By default, all requests are taken from
this directory, but
# symbolic links and aliases may be used to point to
other locations.
DocumentRoot "C:/Swarm-2.1.1"
```
(prima correzione)

```
# Each directory to which Apache has access, can be
configured with respect
# to which services and features are allowed and/or
disabled in that
# directory (and its subdirectories).
# First, we configure the "default" to be a very
restrictive set of
# permissions.
<Directory />
    Options FollowSymLinks
    AllowOverride None
</Directory>

# Note that from this point forward you must
specifically allow
# particular features to be enabled - so if
something's not working as
# you might expect, make sure that you have
specifically enabled it
# below.
# This should be changed to whatever you set
DocumentRoot to.
<Directory "C:/Swarm-2.1.1">
```
(seconda correzione)

4.4.2 Swarm

Swarm, come detto, è nato nel ’95 al Santa Fe Institute (USA). L'obiettivo era creare un insieme di programmi e librerie standard, da usare per simulare ed analizzare sistemi complessi di comportamento, nell'ambito delle scienze naturali e sociali.

L'idea era di avere uno strumento che permettesse ai ricercatori di sviluppare le loro applicazioni senza doversi preoccupare di risolvere problemi tipici di programmazione informatica (interfaccia utente, salvataggio ed archiviazione dati, gestione dell’ambiente grafico, ecc.). Inoltre, Swarm fornisce una struttura per simulazioni ed un linguaggio comune che le rende accessibili agli altri ricercatori.

Verifica delle impostazioni di Swarm:

Nella directory *c:\Swarm\bin* aprire il file *jdkcswarm* e verificare nella seguente riga di comando che la versione di jdk indicata corrisponda a quella installata.

```
JDKPATH=`regtool get
'HKLM\SOFTWARE\JavaSoft\Java Development
Kit\1.3\JavaHome' 2>/
```

La versione di default indicata è la 1.2, nel nostro caso, quindi, l’abbiamo sostituita con la 1.3.

Nella stessa directory aprire il file *jdkswarm* e verificare nella seguente riga di comando che la versione di jdk indicata corrisponda a quella installata.

```
JDKPATH=`regtool get
'HKLM\SOFTWARE\JavaSoft\Java
Development Kit\1.3\JavaHome' 2>/
```

La versione di default presente è la 1.2, nel nostro caso quindi l’abbiamo sostituita con la 1.3.

4.4.3 Java 2 Sdk

Il software Java 2 Sdk è un ambiente di sviluppo per la costruzione di applicazioni, *applet* e componenti che possono essere sviluppate in una piattaforma Java. Tale software include strumenti utili per l'elaborazione e il collaudo di programmi scritti nel linguaggio di programmazione Java. Questi strumenti sono stati progettati per essere utilizzati dalla riga di comando. Solo l'*appletviewer* possiede un'interfaccia grafica.

4.4.4 Autoexec.bat

Nel file *Autoexec.bat* è necessario verificare la presenza della seguente riga di comando;

```
SET PATH=c:\Programmi\jdk1.3.0_02\bin;c:\swarm-
2.1.1\sum10
```

con jdk installato nella directory *c:\Programmi\jdk1.3.0_02*

4.5 Sequenza dei comandi per l'avvio dell'esperimento

1) Lanciare il programma Apache (comparirà una finestra dos che servirà, passivamente, per l'esecuzione dell'esperimento)
2) Dal terminale di Unix, nella *directory* che contiene i file in Java dell'esperimento, compilare gli stessi con la seguente istruzione:
   *jdkcswarm *.java*
3) (Se la versione di jdk è inferiore alla 1.3 dal *prompt* di dos è necessario inserire nella *directory* dell'esperimento il comando *rmic*)
4) Dal *prompt* di dos, nella *directory* dell'esperimento, inserire il comando

start rmiregistry 1098

5) Dal terminale Unix lanciare

jdkswarm "nome dell'esperimento"

4.6 Swiee e gli esperimenti sulla rete

Il progetto Swiee utilizza la tecnologia TCP/IP tipica dell'internet per creare le connessioni tra *client* e *server*. L'intero software è stato studiato per poter essere utilizzato in "laboratorio", luogo in cui è possibile controllare facilmente tutte le variabili ambientali. E', però, possibile estendere il laboratorio al mondo intero utilizzando la rete internet. La scelta di questo strumento, per molti aspetti eccellente, ha, però, sia vantaggi che svantaggi, eccone alcuni.

- Vantaggi:
  - Consente una notevole estensione dell'universo coinvolto nell'esperimento simulativi
  - Si ha una maggiore validità "ecologica" dell'esperimento in quanto l'accesso alla sperimentazione è immediato, non sono quindi presenti filtri e barriere psicologiche in cui ci si può imbattere nel caso in cui l'ambiente sia per esempio una classe
  - Non sussistono problemi di tempo e la partecipazione è più confortevole e meno costosa
  - E' più semplice raggiungere gruppi appartenenti a diverse classi demografiche
  - La gestione dell'esperimento è molto meno costosa

- Svantaggi:
  - I soggetti dell'universo raggiunto sono molto più segmentati sia per fascia di reddito che per età

- E' complicato controllare l'ambiente in cui si sta operando, in particolare: l'accesso alle informazioni non può essere gestito come in laboratorio; è difficile verificare con assoluta certezza l'identità dei soggetti; si corre il rischio che alcuni individui abbandonino l'esperimento durante lo svolgimento.

Tuttavia l'aspetto fondamentale da controllare con particolare cura nello svolgimento di esperimenti che si basano sulla rete è l'insieme delle variabili ambientali. Lo stesso sperimentatore, nel momento in cui decide di sfruttare l'internet per i propri studi, è consapevole dei problemi e dei rischi cui va incontro e proprio per tali motivi svilupperà il software nel modo più accurato possibile per limitare al massimo le distorsioni e gli eventuali svantaggi che da tale approccio derivano.

Ovviamente lo sviluppo di un'applicazione che, per mezzo dell'internet, consenta una efficace sperimentazione richiede un investimento iniziale molto maggiore di quello necessario per la sperimentazione in laboratorio, ma è indispensabile ricordare come questi due differenti approcci non siano sostitutivi l'uno dell'altro, ma piuttosto complementari.

PARTE QUINTA

Uno sguardo alla letteratura: i Classroom Game

5 Classroom game

Anche negli Stati Uniti l'economia sperimentale svolge un ruolo molto importante, ma a differenza di quanto accade in Italia, viene utilizzata anche per fini didattici e non solo come strumento di ricerca.

L'economia è infatti spesso insegnata ad un livello di astrazione che può impedire ad alcuni studenti di raggiungere intuizioni basilari e grazie all'utilizzo dei *classroom game*, affiancati alle lezioni classiche, è possibile permettere agli allievi di interagire, incrementandone l'interesse e diminuendo lo scetticismo verso le teorie economiche. In questo capitolo presenterò una breve descrizione di alcuni giochi di classe; tali esperimenti sono molto più complicati di quelli svolti da noi in quanto hanno appunto come obiettivo la spiegazione di teorie complesse, anche se vengono svolti senza l'impiego di strumenti particolarmente sofisticati.

5.1 Una "cascata" di informazioni

Sebbene la previsione economica sia tipicamente discussa da una sola persona, un esercizio interattivo è potenzialmente più interessante per gli studenti. Questo esercizio è scaturito da situazioni nelle quali le persone devono mettere insieme informazioni proprie e informazioni che si intuiscono dal comportamento di altri. Per esempio è possibile che un datore di lavoro decida di non assumere un individuo, anche se molto promettente, perché è venuto a conoscenza del fatto che lo stesso lavoratore è stato più volte respinto in seguito a domande di lavoro effettuate nello stesso settore. In questo caso l'informazione negativa derivante da esperienze precedenti potrebbe prevalere sulle ottime impressioni destate nel datore di lavoro. Allo stesso modo effetti di *"bandwagon"* (convoglio) si possono verificare in settori quali la politica, le decisioni di consumo, l'adozione di nuove tecnologie di produzione: osservazioni relative a informazioni private vengono desunte dall'analisi delle decisioni prese dagli altri individui.

Il gioco intitolato "Una cascata di informazioni" è un esperimento di previsione sequenziale nel quale gli studenti, così come il precedente datore di lavoro, ricevono

informazioni private semplici, proprio come un intervistatore presente ad un colloquio di lavoro, e fanno predizioni circa un evento sconosciuto (esempio: abilità e capacità del richiedente). Le previsioni sono pubbliche e sequenziali in modo che gli studenti che operano in un momento successivo possano utilizzare le decisioni prese dai predecessori per fare delle previsioni circa le informazioni private di chi ha giocato prima. Un modello di conformità può nascere se le previsioni iniziali coincidono e se le informazioni predette dominano sulle informazioni private degli studenti che prendono le decisioni in fasi successive del gioco. Chi segue è d'accordo con una predizione "universale" anche se potrebbe rivelarsi una previsione errata, basata solo sulle prove raccolte. Questo tema è conosciuto come "informazione a cascata" (Bikhchandani, Hirshleifer e Welch, 1992).

Questo esercizio può essere utilizzato in modo efficace se presentato in unione con le teorie relative all'informazione, all'incertezza e con la teoria dei giochi in corsi di economia preliminari o intermedi. E' un valido strumento anche per corsi quali economia statistica, finanza e economia manageriale, nonché per ambiti quali le scienze politiche, la psicologia e la sociologia.

L'esercizio può durare dai 30 ai 45 minuti a seconda della dimensione della classe alla quale viene proposto. Per effettuare l'esperimento sono necessari: un mazzo di carte, tre biglie identiche di un colore e altre tre, sempre perfettamente uguali, di un altro colore (si possono impiegare per esempio palline da golf o da pingpong), tre coppe di plastica opache. Si supponga di avere tre biglie scure e tre chiare e si mettano delle etichette su due delle tre tazze con le scritte rispettivamente "urna A", "urna B". L'urna A contiene due palline nere e una bianca, l'urna B contiene due palline bianche e una nera. L'urna A possiamo immaginare corrisponda ad un'elevata abilità nell'ambito lavorativo e la pallina nera potrebbe rappresentare un'intervista favorevole, ma è meglio rimandare le interpretazioni ad una fase successiva in cui le decisioni siano già state prese.

Per iniziare è necessario mostrare agli studenti il contenuto delle diverse urne e dire loro che si può scegliere indifferentemente tra le varie urne. Innanzitutto, con il lancio di una moneta, si decide quale delle urne (A o B) svuotare, per riversarne il contenuto nella terza priva di etichetta. Ovviamente gli studenti non devono sapere quale dei due contenitori è stato svuotato. Terminata la fase di preparazione si fanno

avvicinare gli studenti all'urna priva di etichetta, per visualizzarne velocemente il contenuto. Ogni giocatore fa così una previsione pubblica relativa a quale delle due urne sia stata svuotata (A o B) e gli studenti che seguono possono utilizzare queste informazioni. Poiché gli studenti prendono le proprie decisioni in sequenza è necessario stabilire un ordine in cui tali decisioni verranno prese. Se la classe è piccola (da 6 a 10 persone) è più opportuno adottare un ordine casuale, ma è anche possibile utilizzare il mazzo di carte per stabilire l'ordine con cui operare. Se la classe è composta da più individui è possibile adottare lo stesso metodo suddividendo gli studenti in piccoli gruppi per ogni sequenza di decisioni. Una volta selezionati gli studenti bisogna far loro estrarre, in disparte, una biglia dal recipiente privo di etichetta e chiedere loro di prevedere quale urna (A o B) sia stata svuotata e poi annunciare la previsione, ma senza indicare le caratteristiche della biglia estratta, alla classe intera. Il procedimento viene ripetuto finché tutti gli studenti non hanno estratto una pallina dall'urna. Al termine dei turni viene svelato alla classe quale urna era stata effettivamente svuotata per permettere a ciascuno di valutare la propria previsione. Anche in questo esperimento è utile la distribuzione di un premio per chi ha effettuato la corretta previsione per incentivare le cavie a riflettere prima di dichiarare la previsione. Per un regolare svolgimento del gioco è opportuno effettuare più sequenze di scelta con almeno 5 decisori per ognuna (in classi piccole, 10 in classi con più di 50 individui).

Passiamo ora all'analisi di un possibile svolgimento del gioco appena descritto. La situazione iniziale è indicata nella tabella seguente:

Urna	**Colore pallina**	**Colore pallina**	**Colore pallina**
Urna A	Nero	Nero	Bianco
Urna B	Nero	Bianco	Bianco

Se il primo decisore predice A, gli altri studenti possono immaginare che abbia estratto una pallina nera. Poiché due delle tre biglie nere sono nell'urna A la probabilità di A dopo che la prima estrazione ha indicato una pallina scura è 2/3. Così se la prima persona predice A e la seconda estrae una pallina nera, quest'ultima si trova di fronte all'evento: "due palline scure", lo studente deve dunque razionalmente predire A dal momento che dopo due estrazioni la probabilità che si tratti dell'urna A sono molto elevate. Al contrario nel caso in cui la seconda pallina estratta sia bianca, non si possono

trarre informazioni utili dalla prima estrazione: le probabilità sono identiche e pari ad 1/2. Gli altri studenti possono però a questo punto immaginare che se il secondo giocatore ha predetto B è perché ha estratto una pallina bianca. Ovviamente nel caso in cui i primi due studenti abbiano previsto A - estraendo entrambe una pallina nera - queste previsioni diventano più importanti dell'informazione privata della terza persona. Così via per gli studenti successivi: le informazioni desunte dai primi due prevalgono su qualunque esperienza personale. Questo meccanismo è noto come "informazione a cascata". Gli errori possono però facilmente minare la validità della cascata: un soggetto che valuti più l'informazione desunta personalmente delle previsioni dei compagni può rompere il fragile equilibrio.

Gli esperimenti svolti da Anderson e Holt (1995) mostrano come le cascate si formino nell'80% dei casi. Un esempio lampante di cascata è il seguente: - premesso che l'urna svuotata è la B - il primo giocatore pesca una pallina nera e predice A, il secondo ne estrae una bianca e dichiara B, il terzo una bianca e continua a dichiarare B il quarto una nera, ma basandosi sulle informazioni dei compagni continua a predire B innescando così la cascata che continua per i turni successivi. Una cascata "inversa" può innescarsi nel caso in cui, scelta per esempio l'urna A, i primi due giocatori estraggano una pallina bianca (meno probabile) e dichiarino di conseguenza B. Gli altri razionalmente sceglieranno B qualunque pallina abbiano estratto innescando la cascata inversa. L'introduzione di contenuti di carattere maggiormente economico può essere effettuata con la semplice variazione del *set up* di base del gioco.

5.2 Approvvigionamento volontario di un bene pubblico

L'approvvigionamento di un bene pubblico è un dilemma sociale che viene universalmente proposto durante i corsi di microeconomia. Un semplice gioco di carte può essere utilizzato per proporre agli studenti il conflitto tra gli incentivi individuali ad effettuare una scelta privata e gli incentivi sociali a contribuire ad accantonare scorte di un bene pubblico. Questo esperimento propone una via molto semplice per motivare discussioni sull'altruismo, su strategie per incrementare i fondi privati e per discutere sul ruolo del governo nella risoluzione dei problemi relativi ai beni pubblici. Il gioco di

carte che descriveremo può essere utilizzato in varie categorie di classi, incluse quelle di istituzioni di economia, economia applicata e teoria dei giochi.

In pochi minuti è possibile distribuire le carte e spiegare le regole del gioco. Il gioco inizia con la distribuzione agli studenti del foglio contenente le regole e di quattro carte, due rosse (cuori e quadri) e due nere (fiori e picche). Ad ogni studente verrà chiesto di giocare due carte che andranno depositate sulla cima della pila di carte che stanno sul palmo della mano del professore. I giocatori guadagnano 4$ per ciascuna delle carte rosse in loro possesso e 1$ per ogni carta rossa messa sul mazzo dallo studente stesso o dagli altri. Il guadagno è quindi di 4$ per ogni carta rossa posseduta più 1$ per il totale del numero delle carte rosse che il professore ha collezionato. I guadagni possono essere ipotetici, ma al termine dell'esercizio è opportuno estrarre a sorte un allievo al quale dare almeno una parte proporzionale dell'intero guadagno.

Un bene pubblico puro, come la difesa nazionale, implica sia la non competizione – il consumo del bene da parte di un individuo non limita i benefici di questo bene per gli altri individui - che la non esclusione – è impossibile fare in modo che un individuo non goda dei benefici di questo bene, sebbene non abbia pagato nulla per il suo approvvigionamento -. E' essenziale notare che se si gioca una carta rossa si apporta un beneficio alla collettività. Le carte nere non producono guadagni e permettono di mantenere privata la decisione dell'individuo. Questo meccanismo permette agli studenti di giocare due carte coperte senza far intendere quante carte rosse abbiano effettivamente giocato. L'intero processo viene ripetuto per una serie di turni, al termine del quale si ritirano le due carte possedute da ogni studente, si conta il numero totale delle carte rosse giocate, si annuncia questo numero e si restituiscono le carte. Al termine del conto, le carte vanno restituite ai giocatori in modo tale che ciascuno si ritrovi esattamente con le stesse che possedeva in partenza. Dopo ogni *round* gli studenti devono registrare il proprio guadagno.

La ripetizione del gioco consente innanzitutto di imparare a giocare e di individuare nuove strategie. Dopo alcuno turni può essere interessante modificare leggermente le regole del gioco variando, per esempio, il valore di una carta rossa (da 4 a 2$); questa semplice modifica nella maggior parte dei casi implica un incremento dei contributi "versati". Si può anche decidere di permettere un minimo di conversazione tra i giocatori (tra il decimo e l'undicesimo turno), questa conversazione non vincolante

può consentire agli studenti di riflettere con più attenzione sul problema e può consentire una maggior coordinazione nel raggiungimento di un più elevato livello di contribuzione. Questo tipo di esercizio può essere svolto con un numero di studenti compreso tra cinque e tredici per ogni mazzo di carte. Se la classe è composta da un numero più elevato di individui è opportuno suddividerli in più gruppi. E' inoltre possibile svolgere due esperimenti nella stessa aula avendo così la possibilità di confrontare i risultati al termine dell'esercizio.

Williams e Walker[19] (1993) descrivono un esercizio effettuato con l'ausilio di computer nel quale gli studenti prendono le proprie decisioni fuori dalla classe e durante più settimane. Questo modello di analisi consente di avere una maggiore certezza circa le decisioni prese nel caso in cui la classe sia composta da un numero molto elevato di elementi.

Sono possibili numerose varianti all'esperimento. Una di queste può introdurre la possibilità per i giocatori di ritrattare le proprie decisioni di contribuzione. Molte altre modifiche sono proposte da Hoaas e Madigan[20] (1996).

Un buon punto di partenza per la discussione in classe relativa al risultato degli esercizi è l'utilizzo di argomenti che emergono durante il periodo di comunicazione concesso nel corso dell'esperimento stesso. Spesso gli studenti riconoscono sia l'incremento di guadagni che è raggiungibile attraverso un maggiore coordinamento, sia gli incentivi al *free-ride*. Generalmente i giocatori sono disponibili a discutere relativamente a decisioni e strategie adottate. Sovente emerge che le scelte adottate sono dettate da forme di generosità e altruismo e dalla convinzione di poter incoraggiare anche gli altri partecipanti ad una maggiore cooperazione. Ovviamente al termine della discussione è opportuno trarre alcune conclusioni spiegando per esempio che se in un singolo turno l'individuo può massimizzare i propri guadagni monetari non contribuendo al bene della collettività (tenendo quindi in mano le due carte rosse), è altrettanto vero che i guadagni per l'intero gruppo vengono massimizzati solo nel caso in cui ciascuno contribuisca pienamente. La discussione può anche essere focalizzata sulle caratteristiche del bene pubblico (non escludibilità e non competitività).

[19] "Computerized Laboratori Exercises for Microeconomics Education: Three Applications Motivated by the Methodology of Experimental Economics".
[20] "The alleviation of Free-Riding: A Research Program Progress Report".

Nella teoria del gioco di classe le discussioni possono anche essere centrate sulla risoluzione dell'equilibrio di Nash e su come i cambiamenti nei parametri dell'esercizio (ad esempio il valore della carta rossa) possano o meno far variare l'equilibrio delle contribuzioni. Proprio il cambiamento del valore di una carta rossa può stimolare la discussione sui diversi costi di contribuzione. Le persone appartenenti ai differenti ceti sociali possono avere, per esempio, diversi incentivi di contribuzione, ed è importante notare come per contribuzione si intenda non solo quella prettamente monetaria, ma anche quella legata al fattore tempo (esempio: volontariato).

Da questo gioco emerge sicuramente l'importanza fondamentale del fattore contribuzione e dell'aspettativa che anche gli altri individui siano interessati al benessere pubblico. I contributi, infatti, non sono in contrasto con la teoria economica e nemmeno con la razionalità, ma essenziale è che tale convinzione sia diffusa tra i molteplici operatori economici.

5.3 Comprendere la regola di Bayes

La regola di Bayes è da sempre uno strumento molto utile nell'analisi dei dati economici. La sua importanza è recentemente cresciuta a seguito degli studi sui mercati con informazioni asimmetriche o con incertezza riguardo a salari e prezzi.

Il *classroom game* che permette di comprendere tale regola dura circa trenta minuti e può essere effettuato con l'utilizzo di tre palline di un colore, tre di un altro colore, un dado a sei facce e due identiche tazze di plastica. Come nel primo gioco esaminato, anche in questo è possibile utilizzare palline da golf o da pingpong che garantiscono identiche dimensioni e l'indeformabilità. Ogni tazza contiene due biglie di un colore ed una dell'altro (la tabella seguente schematizza la situazione iniziale).

Urna	**Colore pallina**	**Colore pallina**	**Colore pallina**
Urna A	Rosso	Rosso	Bianco
Urna B	Rosso	Bianco	Bianco

Un volontario si nasconde (magari dietro ad uno schermo) e lancia il dado. L'urna A sarà utilizzata se il risultato del lancio è 1, 2, 3, la B negli altri casi. Gli studenti della

classe non devono, però, vedere il risultato del lancio ed è opportuno riassumere sulla lavagna le regole come mostrate nella tabella precedente. In seguito viene estratta una biglia e mostrata alla classe; ciascuno studente deve scrivere su di un foglio intestato con nome e cognome la probabilità che la pallina sia stata estratta dall'urna A e una breve spiegazione di come abbia raggiunto la propria conclusione. Il coinvolgimento dei vari studenti assicura una molteplicità di risposte differenti e la spiegazione indicata assicura che le spiegazioni non corrette vengano convenientemente dimenticate. E' opportuno consegnare a ciascun giocatore un foglio che oltre a riportare la tabella sopra descritta ponga le seguenti domande:

1. Dopo che la tazza è stata selezionata dal lancio del dado, ma prima di vedere l'estrazione, qual è la probabilità che sia stata utilizzata l'urna A?
2. Il colore della prima biglia estratta è …… qual è la probabilità che sia stata utilizzata l'urna A? Motivare brevemente la risposta.
3. La prima biglia è stata rimessa nell'urna ed è stata estratta una seconda pallina dalla stessa urna, il colore della seconda biglia estratta è …… qual è la probabilità che sia stata estratta dall'urna A? Motivare brevemente la risposta.

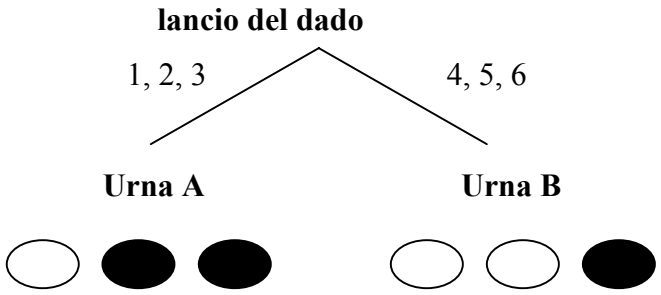
Al termine di questa fase vengono raccolti i fogli e in una tabella vengono riportati i risultati delle risposte. Se la prima estrazione ha mostrato una pallina rossa, la probabilità più riportata dagli studenti che indica la provenienza dall'urna A è compresa tra 1/3 e 2/3. Riportare l'insieme delle probabilità indicate permette di stimolare la discussione e di arrivare ad una versione intuitiva della regola di Bayes.

5.3.1 Un calcolo euristico

Si supponga che la prima estrazione abbia mostrato una biglia rossa, agli studenti viene richiesto di indicare la probabilità che sia stata utilizzata l'urna A. La risposta comune alla maggior parte degli studenti – laureati inclusi – è 1/2. Quando si chiede ad uno studente perché abbia scritto 1/2 potrebbe rispondere che ciascuna urna aveva la medesima probabilità di essere selezionata. Nel caso in cui, invece, nessuno abbia risposto così è importante chiedere perché non abbiano indicato 1/2 dato che ciascuna urna aveva la stessa probabilità di essere estratta a sorte. E' utile iniziare con questa

domanda relativamente facile per far prendere fiducia e confidenza col gioco agli studenti e per stimolare una successiva discussione su argomenti più interessanti e complicati. Un'altra risposta scorretta, ma spesso riscontrata, relativa sempre alla probabilità che l'urna in questione sia la A è 1/3. Se si chiedono motivazioni relativamente alla affermazione indicata, spesso si ottiene come risposta che la probabilità di aver utilizzato l'urna A è di 1/2 e la probabilità di aver estratto la pallina rossa da tale urna è di 2/3, perciò la probabilità che l'estrazione della pallina rossa derivi dall'urna A è di 1/2 per 2/3 e cioè di 1/3.

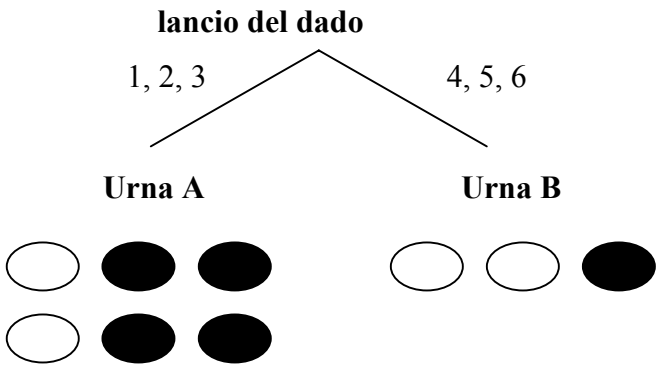
La migliore risposta che si può dare ad una affermazione del genere è la domanda: allora qual è la probabilità che sia stata utilizzata l'urna B? Oppure: seguendo il medesimo ragionamento, se la probabilità che la pallina derivi dall'urna B è di 1/2 per 1/3 e cioè di 1/6 dove va a finire la regola fondamentale delle probabilità (la somma delle varie probabilità che si verifichino i diversi eventi deve è sempre pari a 1)? Il risultato corretto è ovviamente una probabilità pari a 2/3 che la biglia rossa sia stata estratta dall'urna A e 1/3 che provenga dalla B. Se qualcuno, invece, avesse correttamente indicato 2/3 per l'urna A è importante dare lettura delle motivazioni indicate e chiedere ulteriori spiegazioni; senza preoccuparsi che la risposta provenga da un corretto calcolo delle probabilità o da un fortunato tentativo, è bene focalizzare l'attenzione di tutti sulla figura seguente:



e chiedere se esiste un metodo semplice per contare le biglie e per determinare la probabilità che la biglia rossa estratta derivi dall'urna A. Qualcuno potrebbe realizzare, con deboli insinuazioni e argomentazioni, che le sei palline hanno la stessa probabilità di essere estratte prima che il dado venga lanciato e che due delle tre palline rosse si

trovino nell'urna A e di conseguenza che la probabilità successiva che l'urna A dia una pallina rossa è di 2/3.

Supponiamo ora che la pallina rossa estratta venga riposizionata nella tazza e che venga detto agli studenti che la seconda estrazione verrà fatta dal medesimo contenitore selezionato col lancio del dado. Avendo già rilevato l'estrazione di una biglia rossa gli studenti sono portati a credere, prima della seconda estrazione, che la probabilità che sia usata l'urna A sia di 2/3. Nonostante il numero delle palline contenute nelle rispettive urne non sia mutato, è possibile rappresentare questo modo di pensare supponendo che l'urna A abbia il doppio del numero di palline racchiuse nell'urna B e che le biglie delle due urne abbaino la stessa probabilità di essere estratte. Questo modo di pensare è rappresentato nella figura successiva:



Le proporzioni di biglie bianche e scure (rosse) sono mantenute costanti. Se dopo l'estrazione della pallina rossa la situazione è quella rappresentata nella figura, è naturale che una biglia bianca, al momento della seconda estrazione, ha la medesima probabilità di provenire da un'urna come dall'altra (ciascuna urna contiene ora due sfere bianche). La probabilità di estrarre al secondo turno una biglia bianca è ora pari ad 1/2. Se invece si suppone che le prime due estrazioni abbiano mostrato due sfere rosse; come prima, la situazione che si presenta dopo la prima estrazione può essere rappresentata dalla figura vista sopra. Dal momento che quattro delle cinque palline rosse sono nell'urna A, dopo la seconda estrazione (sempre biglia rossa) la probabilità dell'evento pallina rossa è pari a 4/5.

Una volta spiegato agli studenti come calcolare esattamente le probabilità che si manifestino gli eventi rappresentati nella prima figura è necessario effettuare nuove

estrazioni e valutare nuovamente il livello di apprendimento raggiunto. Anche in questo caso in assenza di un incentivo di carattere economico il numero di errori è molto maggiore rispetto a medesimi esperimenti svolti sfruttando uno stimolo quale potrebbe essere una compensazione monetaria.

5.4 L'interazione strategica sull'internet

Questo gioco può essere effettuato individualmente o in gruppo sull'internet e possono parteciparvi studenti di differenti Facoltà o addirittura appartenenti a diverse nazioni. Il programma su cui è basato tale esperimento è scaricabile gratuitamente all'indirizzo www.cmu.edu/comlabgames. E' stato sviluppato da Grobelnik, Kese, Miller e Prasnikar e può consentire la gestione di un numero molto elevato di partecipanti. Ha una ampia varietà di opzioni e prevede l'operatività di un programma moderatore. Gli studenti che decidono di partecipare al gioco possono utilizzare una connessione standard ad internet e Netscape o Internet Explorer come *browser.* Il programma permette all'istruttore di creare un gioco specifico indicando le caratteristiche e il numero di decisioni che ciascun giocatore deve prendere, la matrice relativa ai premi, il protocollo per creare gli abbinamenti tra i giocatori (casuale o tramite un preciso algoritmo) e un protocollo necessario a fermare il gioco. Tutte queste caratteristiche vengono salvate sul computer dell'istruttore e possono essere modificate in un qualsiasi momento. Quando il gioco viene fatto iniziare, il programma moderatore invia un messaggio contenente il suo indirizzo internet e i giocatori, avendo a disposizione tale indirizzo, possono connettersi e interagire.

Ciascun gioco consiste in una serie di turni in cui i giocatori devono prendere una serie di decisioni in modo indipendente. All'inizio del gioco a ciascuno studente viene assegnato un preciso ruolo e dopo ogni turno vengono visualizzate, sui rispettivi terminali, le scelte di tutti gli operatori; un file di *output* registra le mosse degli individui e un software statistico effettua numerose analisi.

Descriviamo ora un gioco per due individui che può essere interpretato sia come un mercato di Cournot, sia come un problema di risorse di *common-pool,* ma focalizzeremo maggiormente l'attenzione sul secondo tipo di interpretazione, meno

familiare a istruttori e studenti. La differenza principale nei due tipi di analisi risiede nel fatto che in un problema di *common-pool* l'accordo migliore che due giocatori possono raggiungere coincide anche con il miglior risultato ottenibile a livello sociale, mentre nel caso del mercato di Cournot è l'accordo fraudolento a fornire il massimo profitto per i venditori, a svantaggio della popolazione di consumatori. Se prendiamo, ad esempio, la pesca come risorsa di *common-pool* allora il numero di scelte relative a ciascun giocatore corrisponde al livello dello sforzo profuso nell'attività. Assumiamo che i due giocatori abbiano le medesime capacità e definiamo lo sforzo impiegato dal giocatore *i,* x_i e lo sforzo totale X; il giocatore *i* avrà una porzione x_i / X di catture. La seconda proposizione utilizzata per costruire la matrice dei pagamenti implica che il valore totale delle catture sia una funzione quadratica dello sforzo totale, $AX-BX^2$. Se il costo dello sforzo è una funzione lineare del livello di sforzo stesso, Cx_i, allora il pagamento del giocatore *i* è dato da $(x_i/X)(AX-BX^2)-Cx_i$, oppure analogamente, $x_i(A-BX)-Cx_i$. Questa struttura dei pagamenti corrisponde a quella di un mercato di Cournot con un costo marginale costante e una funzione di domanda inversamente lineare.

Un set di valori per i parametri, utilizzato con successo in giochi di classe, deriva dagli studi di Holt (1985)[21]: A=30, B=1, C=6 ed un valore costante pari a 45 che va aggiunto a tutti i pagamenti per evitare di rilevare la prevalenza di *payoff* negativi. Questi parametri consentono di generare la matrice dei pagamenti. Il pagamento relativo a ciascun giocatore è determinato dall'intersezione riga/colonna (decisioni dei due giocatori) e il pagamento del giocatore "di riga" è indicato per primo. In base ai valori presenti in matrice sarà dunque possibile individuare i livelli di sforzo ottimali che i due giocatori dovrebbero scegliere per massimizzare i rispettivi *payoff*.

Il gioco relativo alle risorse di *common pool* può essere spiegato con la distribuzione ai partecipanti di un foglio contenente le seguenti regole:

> "Questo esercizio comporta una situazione per cui dovrai interagire con un'altra persona e ciascuno dovrà decidere quante ore dedicare alla pesca. Sforzi maggiori sono più costosi per te, ma un aumento del tuo impegno incrementerà la tua *share*. Gli effetti relativi ai fattori costo e catture saranno sommati e riportati nella matrice dei pagamenti che potrai visualizzare sul

[21] "Classroom Games: A Prisoner's Dilemma".

tuo schermo. Il tuo sforzo sarà rappresentato con un numero compreso tra 4 e 22 e la matrice dei pagamenti indicherà il tuo guadagno, dato l'impegno profuso nell'attività dalla coppia. Nel momento in cui ti connetterai per iniziare l'esercizio, sarai designato come giocatore di riga o di colonna (la tua scelta sarà quindi rappresentata su una riga e quella del tuo partner su una colonna). Il tuo pagamento sarà il numero visibile nel punto di intersezione tra la riga e la colonna e verrà colorato di rosso. Il pagamento dell'altro giocatore è riportato in nero. Per esempio: se entrambe i giocatori scelgono di applicarsi con uno sforzo pari a 14 allora entrambi guadagneranno 17; oppure se tu scegli un livello di sforzo pari a 10 e l'altro giocatore sceglie 4, il tuo guadagno sarà pari a 95 mentre il tuo compagno ricaverà 45. Entrambi prenderete delle decisioni senza sapere quale sarà la scelta dell'altro giocatore in modo che dobbiate riflettere sulle possibili scelte che l'altro individuo effettuerà. Quando tutti e due avrete scelto il calcolatore procederà con il calcolo dei guadagni e li visualizzerà sul terminale. Il programma memorizzerà tutte le tue decisioni e i tuoi guadagni e l'istruttore ti dirà quale sarà il numero dei turni e se e quando cambierai partner".

Al termine dell'esperimento sarà poi possibile, sfruttando i dati raccolti dal calcolatore, analizzare tali valori e effettuare discussioni, chiedendo agli studenti di motivare le proprie scelte, rilevando gli equilibri raggiunti e quelli potenzialmente raggiungibili con scelte differenti.

5.5 Un "mercato dei limoni"

Se la qualità di un prodotto non può essere osservata da un eventuale compratore a priori per poter decidere la scelta, allora i venditori saranno indotti a sfruttare a loro guadagno tale situazione. I compratori diventeranno allora riluttanti a pagare prezzi elevati aspettandosi prodotti di scarsa qualità (bidoni). La definizione "mercato dei

limoni" deriva da Gorge Akelrof (1970)[22], il quale ha spiegato come la pressione determinata dalla competizione possa causare il deterioramento della qualità a livelli talmente bassi che il mercato potrebbe addirittura fallire. Numerosi corsi di economia trattano, anche approfonditamente, questo argomento.

Gli incentivi che nascono nei mercati con asimmetrie informative emergono chiaramente nel successivo *classroom game*.

Gli studenti venditori scelgono sia il livello qualitativo che il prezzo a cui vendere i propri prodotti. Inizialmente sia prezzo che qualità sono, però, definiti e i compratori possono scegliere tra le varie possibilità. Con un *setup* basato sull'informazione completa i prezzi e la qualità raggiungono velocemente i livelli di efficienza che massimizzano il surplus totale. Successivamente, con l'introduzione delle asimmetrie informative, qualità e prezzi cadranno a livelli molto bassi. I risultati che si raggiungono con questo genere di esercizio consentono di introdurre dibattiti relativi ad asimmetrie informative, fallimento dei mercati e sui possibili rimedi (standard di qualità).

Per incominciare correttamente l'esercizio è indispensabile riprodurre una tabella sulla lavagna. Tale tabella deve avere una colonna per ciascun venditore, col numero relativo al venditore scritto in cima e con un numero sufficiente di righe da permettere lo svolgimento di almeno 6 turni. Ciascuna cella della tabella sarà utilizzata per indicare il prezzo, il livello di qualità e il numero di unità vendute da ciascun venditore in ciascun periodo.

[22] "The Market for lemons: Quality Uncertainty and the Mechanism"

	Venditore 1	Venditore 2	Venditore 3
Tempo 1 (informazione completa)	$11.50 livello 3 1 unità	$6.00 livello 2 2 unità	$12.00 livello 3 1 unità
Tempo 2 (informazione completa)	$5.75 livello 2 2 unità	$5.50 livello 2 1 unità	$1.90 livello 1 1 unità
Tempo 3 (informazione completa)	$5.65 livello 2 1 unità	$5.60 livello 2 2 unità	$5.60 livello 2 1 unità
Tempo 4 (solo informazione di prezzo)	$2.40 livello 1 1 unità	$5.60 livello 2 1 unità	$2.40 livello 1 2 unità
Tempo 5 (solo informazione di prezzo)	$2.40 livello 1 1 unità	$1.65 livello 1 1 unità	$5.50 livello 1 2 unità

L'esercizio inizia con la suddivisione degli studenti in venditori e compratori e con la spiegazione delle regole del gioco. Un mercato che risulta funzionare particolarmente bene è quello composto da 3 venditori e 4 compratori (oppure 6 venditori e 8 compratori). Un numero più elevato di compratori fa durare troppo i turni di gioco dal momento che i compratori prendono le decisioni di acquisto in sequenza. Per classi più numerose è consigliabile raggruppare nella figura di un compratore più di uno studente. La discussione che probabilmente nascerà dal compattare più persone in un solo operatore di mercato, oltre a stimolare maggiormente la partecipazione e l'interesse, consentirà una migliore e più approfondita comprensione del gioco. Se le classi sono molto numerose è opportuno che un assistente si preoccupi di evitare la comunicazione tra compratori e venditori, magari posizionandoli in stanze separate.

Lette le regole del gioco, è bene lasciare alcuni minuti ai partecipanti per riflettere e per permettere loro di effettuare eventualmente domande. Al termine di questa fase preliminare si apre il mercato, consentendo quindi ai venditori di scegliere i livelli di qualità e prezzo del prodotto per il primo turno. I venditori guadagnano vendendo ad un prezzo maggiore del costo, ma non è permesso vendere più di due unità di prodotto per ciascun turno. Registrate le scelte dei venditori va compilata la tabella (come sopra). Al

termine di questa fase si estrae a sorte il primo compratore e poi si procede in modo, per esempio, orario. Il primo compratore della sequenza deve decidere se e eventualmente da quale venditore comprare, al massimo è possibile acquistare un solo oggetto per ogni *round*. Un compratore guadagna se riesce a comprare ad un prezzo inferiore al valore del bene che sceglie (valore al quale i compratori potrebbero rivendere l'unità). Presumibilmente il venditore deciderà di vendere una prima unità, ma non la seconda se non riuscirà a coprire i suoi costi. Al termine di una transazione si procede con il secondo compratore e così via. Una volta effettuati, gli acquisti vanno registrati nell'apposita tabella e al termine di questa prima fase si prosegue con le nuove decisioni di prezzo e qualità dei venditori. La domanda che si genera sul nostro mercato è dettata dal valore di rivendita conosciuto dai compratori. Ciascun compratore può acquistare al massimo una sola unità dei prodotti in ogni periodo ed il valore del bene acquisito dipende dal suo grado di qualità: $4 per il livello 1, $8.80 per il secondo livello, $13.60 per il terzo. La domanda di un compratore relativa ad un prodotto col primo livello di qualità è quindi perfettamente inelastica per tutti i prezzi inferiori ai 4 dollari. L'offerta del mercato è invece determinata dai costi imposti ai venditori. Ciascun venditore ha a disposizione due unità e il costo della seconda unità e di 1 dollaro superiore rispetto al costo della prima. Per un grado di qualità pari ad uno, il costo della prima e della seconda unità prodotte è di rispettivamente $1.40 e $2.40.

Quando il mercato si stabilizza ad un livello corretto si annuncia ai giocatori un cambiamento. I venditori devono scegliere sempre livelli di qualità e prezzi, ma ora nella tabella verranno indicati solo i prezzi di vendita. I compratori non sapranno quindi il livello qualitativo del bene che si accingeranno a comprare. Questo cambiamento va annunciato, da parte dell'istruttore, in modo da non far intendere che esistono nuove possibilità di guadagno. La procedura del gioco è esattamente identica a quella descritta per la prima fase. Dopo pochi turni, generalmente ne sono sufficienti due, si raggiungerà un equilibrio al più basso livello di qualità, uno, che ovviamente non è il livello ottimale per la società. Da un gioco così strutturato possono emergere notevoli spunti di riflessione per interpretare quanto possa incidere su di un mercato un'informazione asimmetrica e sull'importanza del ruolo che potrebbe svolgere un controllore-moderatore che si occupi della corretta circolazione delle notizie.

5.6 Speculazioni e bolle sul mercato dei capitali

La teoria standard per la valutazione di titolo prevede che tale strumento dovrebbe essere venduto al valore attuale scontato dei profitti che sarà in grado di generare in futuro. Gli scambi avvengono in quanto gli operatori hanno idee differenti circa il rischio e le aspettative sui futuri profitti. Nel testo "General Theory" Keynes (1936) indica come i prezzi dei titoli siano molto spesso determinati non tanto dai fondamentali del mercato a lungo termine, quanto piuttosto dalle aspettative relative alla valutazione che il mercato darà del titolo in questione nell'arco dei 3–12 mesi successivi.

L'esercizio di mercato proposto permetterà un paragone tra i prezzi di compravendita di un titolo e il suo valore reale. L'esercizio prevede di realizzare una contrattazione interattiva per facilitare una discussione circa le aspettative razionali, gli sconti, il comportamento retrogrado di molti operatori e le bolle speculative. L'esercizio può essere molto utile se eseguito in classi di studenti universitari e negli MBA.

In questo esercizio gli studenti commerciano titoli di un determinato valore in una sequenza di *round.* Sono necessari circa 40 minuti per dare lettura delle istruzioni e per effettuare un certo numero di turni. La discussione successiva dovrebbe durare almeno 15–20 minuti. Un massimo di 5 studenti può operare in qualità di commercianti, ma è possibile coinvolgere anche fino a 50–60 individui, a patto che i commercianti siano rappresentati ciascuno da un gruppo di 3 o 5 persone e che gli altri si occupino della registrazione dei dati e degli altri aspetti tecnici.

E' necessario utilizzare 15 fogli di carta colorata per rappresentare i diversi titoli. Ciascun operatore incomincia il gioco con 3 fogli colorati (3 diversi titoli) che possono essere comprati o venduti durante i 10 turni. I commercianti possono guadagnare del denaro dai titoli in tre modi:

1. ciascun capitale posseduto al termine di un turno pagherà un dividendo di un dollaro;
2. è possibile realizzare profitti dalla compravendita di titoli;
3. ciascun titolo ancora esistente al termine dei 10 turni sarà valutato $6.

Non tutti i capitali sopravviveranno fino all'ultimo *round.* Al pagamento del dividendo ($1) c'è una possibilità su sei che ogni titolo posseduto venga distrutto, nel qual caso il

foglio colorato rappresentante il titolo verrà strappato. Per determinare quale capitale distruggere si utilizzerà un dado. Ci si attende che il numero di titoli diminuirà di due durante ciascuno dei primi 3–4 turni e di 1 nei successivi. Qualcuno dei 15 titoli iniziali rimarrà intatto sino al termine del gioco e ciò significa che solo per queste azioni verranno pagati i $6. In termini economici, la proposta di introdurre il rischio che un titolo venga distrutto serve per simulare uno sconto. L'ingrediente analitico dello sconto implica di valutare meno i profitti che ci si attende di ricevere in futuro rispetto a quelli percepiti nel presente. Il gioco riesce a incorporare questo concetto introducendo appunto la possibilità che dopo il pagamento del dividendo l'azione venga distrutta (annullandone quindi il valore futuro). Gli operatori contabilizzano i propri guadagni in apposite tabelle, indicando quindi il numero di titoli e le quantità possedute, i prezzi di vendita e di acquisto delle azioni in modo tale da aver sempre presente la loro situazione. Anche in questo caso l'introduzione di un premio per i vincitori può essere un importante stimolo ad operare al meglio, con impegno e attenzione. La fase di *trading* è condotta secondo il modello dell'asta pubblica e l'istruttore svolge le mansioni del banditore. Ciascun gruppo è definito da una lettera compresa tra la A e la E e un individuo fa da portavoce per i compagni. Per poter immettere proposte in acquisto o in vendita su di un titolo il *trader* deve prima alzare la mano e poi farsi riconoscere. In qualità di banditore l'istruttore deve imporre precise regole di comportamento per garantire l'ordine ed il corretto svolgimento del gioco; inoltre deve iniziare il gioco con una proposta di apertura (es.: il gruppo A offre 700 per l'acquisto di un titolo). Qualche gruppo risulterà più attivo rispetto agli altri. Ciascun'offerta successiva alla prima deve essere maggiore rispetto alla maggiore offerta effettuata e ciascuna domanda deve essere inferiore alle precedenti. Ad un certo punto un gruppo accetterà una qualche domanda/offerta, tutte le altre proposte andranno cancellate e si incomincerà un nuovo ciclo. Durante i primi turni, generalmente più lunghi dei successivi, potranno emergere alcuni dubbi, ma i *round* seguenti non dovranno durare più di 2 minuti; ciò implica l'eventualità che solo due o tre transazione possono essere portate a termine, a seconda della vivacità dei giocatori. E' anche possibile che alcuni turni non vedano il concludersi di alcuno scambio, soprattutto se dopo il fallimento di numerose proposte. Chi si occupa della gestione del tempo ha anche il compito di

avvisare gli operatori del mercato quando mancano 30 secondi allo scadere delle contrattazioni.

Questo esperimento può essere molto utile come base per una discussione di classe sulla valutazione dei titoli. Si può, per esempio, iniziare mostrando un lucido che incorpori i dati relativi alle transazioni effettuate e un grafico che riporti per ogni turno i prezzi a cui si sono effettuati gli scambi per poi riflettere sul *pricing* delle azioni stimolando il dibattito con domande su quali erano le aspettative di prezzo per esempio all'inizio o alla fine del gioco. Come conclusione è opportuno riuscire a far comprendere agli studenti la seguente formula relativa al prezzo di un titolo al tempo t:

$$\text{Prezzo}(t) = 1.00 + [5 / 6] \times \text{Prezzo Atteso } (t + 1)$$

L'andamento dei prezzi delle transazioni avvenute durante l'esperimento può essere paragonato con le predizioni delle aspettative razionali; in particolare gli studenti dovrebbero riconoscere che si genera una circolarità potenziale quando i prezzi correnti dipendono dalle aspettative sui prezzi futuri, ma queste aspettative sono influenzate dai livelli e dal *trend* dei prezzi correnti. Il grafico con in ascissa i periodi e in ordinata il livello dei prezzi può chiaramente mostrare come in alcuni casi si formino bolle speculative anche molto evidenti. Se però le bolle non sono coerenti con le aspettative razionali allora perché anche solo in esperimenti di classe emergono così palesemente? Una risposta risiede nel concetto di irrazionalità competitiva: gli agenti agiscono in modo evidentemente irrazionale in termini di ricavo atteso in un contesto economico, ma è difficile identificare ogni singola azione come irrazionale. Gli agenti che operano sul mercato prevedono che altri *trader* possano avere aspettative di un prezzo maggiore per un determinato titolo oppure che ci siano degli speculatori in procinto di agire. In base a queste aspettative tali individui incominceranno ad acquistare a prezzi ancora bassi, ma la maggiore domanda del titolo implicherà l'incremento di prezzo (necessario per riequilibrare il mercato) e il sorgere di una bolla speculativa.

5.7 Generare moneta

Una parte cruciale della discussione sulla politica monetaria in un corso introduttivo di economia è il ruolo che le banche centrali hanno nella creazione di moneta. Spesso gli studenti trovano tale processo misterioso, anche dopo le tradizionali spiegazioni relative all'espansione dei depositi. Per migliorare la comprensione gli studenti possono essere coinvolti nel processo di creazione di moneta attraverso un *classroom game* nei ruoli di banchieri depositari e prenditori di fondi. L'obiettivo primario dell'esercizio è dimostrare come un iniezione iniziale di risparmi porti ad un aumento più che proporzionale del livello dei depositi. Può anche essere utilizzato come incipit per una discussione circa il ruolo delle banche come intermediari finanziari, l'importanza della fiducia che i depositari hanno sulla copertura delle banche e sul ruolo della banca centrale come strumento di regolazione dell'intero sistema creditizio. E' opportuno che tutti gli studenti abbiano un *background* sufficiente in materia in modo da poter utilizzare una terminologia tecnica senza rischiare di invalidare i risultati del gioco. Questo esercizio coinvolge circa 20 studenti e i benefici diminuiscono sensibilmente in classi con più di 50 individui.

Di seguito verranno indicate, una per una, le procedure che deve seguire l'istruttore per svolgere l'esperimento proposto.

1. Prima dell'inizio della prova ciascun partecipante deve ricevere una copia delle istruzioni del gioco. Inoltre è bene preparare dei dollari, dei titoli di stato e degli assegni personali da distribuire. Si possono utilizzare 10 carte segnate con l'indicazione "$20", 10 segnate con "$20 in Titoli di Stato" e 20 valevoli come assegni. In classe bisogna anche portare due contenitori: uno sarà utilizzato come cassaforte di una generica banca e l'altro come forziere della banca centrale;
2. bisogna reclutare tre volontari: un banchiere, un archivista della banca e un archivista individuale;
3. distribuire i titoli di stato agli studenti che partecipano al gioco;

4. dare letture delle istruzioni e precisare che tutto il denaro è depositato presso la banca (assenza totale di contante) e che tutte le riserve in eccesso saranno prestate;
5. prima che le transazioni vengano portate a termine si chieda agli studenti di calcolare la quantità di denaro (per semplicità la banca inizia priva di depositi). Definire nuovamente i concetti di riserva e il significato della sigla M1;
6. per dare inizio al gioco annunciare che la banca centrale desidera acquistare dei titoli di stato da 5 studenti per un ammontare pari a $100 e che tali titoli verranno pagati in dollari;
7. se necessario si spingano gli studenti a depositare i titoli in cambio di assegni;
8. si istruisca l'archivista individuale a registrare ciascun deposito facendo una lista contenente i nomi degli studenti e il rispettivo ammontare dei depositi detenuti da ciascuno;
9. si istruisca l'archivista della banca a redigere il bilancio della banca; dal punto di vista delle passività i depositi della banca sono aumentati di $100 e anche le riserve sono incrementate di $100 (contante fresco). Queste riserve andranno depositate nella cassaforte;
10. si chieda qual è stato il cambiamento iniziale nell'offerta di denaro. La risposta errata che spesso viene data è $200, infatti la quantità di denaro è data dalla somma di depositi e denaro circolanti al di fuori della banca (quindi $100);
11. data l'iniezione iniziale di riserve, il banchiere sarà ora in grado di elargire dei prestiti. Una lista di domande segnate su un lucido guiderà il banchiere circa le decisioni relative alla concessione o meno di prestiti. Tali domande guida sono:
    - quale percentuale di depositi deve essere mantenuta dal banchiere in riserve? (20%);
    - quale ammontare di riserve devi detenere? (.20 × $100 = $20);

- quanto hai in questo momento in riserve? ($100);
- qual è l'ammontare delle tue riserve in eccesso? ($100 - $20 = $80);
- quanto puoi legalmente prestare? ($80);

12. si ripetano gli *step* da 8 a 11;
13. non è necessario ripetere questa procedura per molti *round*, ma al massimo si può registrare un ulteriore turno in cui .20 × $180 = $36 devono essere tenuti come riserva e i restanti $64 vengono prestati e ridepositati. Si calcoli la quantità totale di denaro ($100 + $80 + $64 = $244) e l'ammontare di riserve nel sistema ($100). Si sottolinei che l'aumento della quantità di denaro e di depositi si riduce progressivamente con ciascun *round* successivo e si chieda perché ciò accade;
14. una volta che gli studenti avranno compreso che le riserve in eccesso disponibili per essere prestate diminuiscono a ciascuno *step,* osserveranno velocemente che tale processo deve ad un certo punto terminare. A questo punto si può mostrare come l'iniezione iniziale di $100 con una percentuale di riserve pari al 20% porterà i depositi e la quantità di denaro ad aumentare fino a $500. Quando i depositi raggiungono i $500 non ci sono riserve in eccesso ($500 × .20 = $100);
15. sebbene possa ora sorgere un gran numero di domande, gli studenti generalmente riescono a comprendere l'attività di generazione di moneta da parte delle banche più facilmente di quanto possano capire la diminuzione della quantità di denaro. Per spiegare tale fenomeno bisogna ipotizzare che la banca centrale abbia ora deciso di vendere titoli di stato per un ammontare di $40. Si cerchi un volontario che sia disponibile ad acquistare due titoli da $20;
16. si faccia compilare allo studente un assegno di $40 intestato alla banca centrale e lo si consegni al banchiere. L'archivista individuale deve segnare tale operazione come decremento del livello dei depositi. Ci si facciano consegnare dal banchiere $40 provenienti

dalle riserve della banca e si metta il contante nel forziere della banca centrale. I $40 in bilancio andranno sottratti sia dalle riserve che dai depositi. Il nuovo livello di riserve è di $60 e la quantità di denaro $460;

17. le domande utilizzate per aiutare il banchiere a prendere le sue decisioni (punto 11) possono essere utilizzate nuovamente per mostrare che le riserve richieste (.20 × $460 = $92) superano le attuali ($60) di $32. Il banchiere deve ora ottenere tale denaro. Il nuovo livello dell'offerta di denaro è $500 - $40 - $32 = $428 e il livello delle riserve è pari a $60;
18. come prima, la diminuzione dell'ammontare di depositi non è sufficiente per ridurre le riserve richieste al livello delle attuali, cosicché bisogna ricercare ulteriori prestiti addizionali. Si ripeta questo processo per due o tre volte per chiarire i dubbi degli studenti;
19. la discussione di classe può essere utilizzata per mostrare che la variazione finale dell'offerta di denaro, dopo aver effettuato un'operazione di vendita titoli dell'ammontare di $40 con una percentuale di riserve richieste del 20%, è di -$200. Si verifichi inoltre che quando i depositi raggiungono i $300 non ci sono riserve in eccesso.

Questo esercizio può durare, in media, tra i 30 e i 40 minuti. Anche in questo caso è bene, terminato il gioco, porre alcune domande provocatorie agli studenti per rimuovere i dubbi rimanenti e per fissare i concetti appresi con lo svolgimento dell'esperimento.

PARTE SESTA

Gli esperimenti svolti con l'aiuto delle simulazioni

6.1 Gli esperimenti di Alessandria

I primi, in Italia, ad organizzare ed effettuare esperimenti svolti con l'aiuto delle simulazioni sono stati i ricercatori dell'Università di Trento, seguiti ben presto da quelli della Facoltà di Giurisprudenza di Alessandria che hanno potuto avvantaggiarsi dall'esperienza dei pionieri veneti. Dal maggio del 2001 anche Torino è diventata sede di esperimenti effettuati in collaborazione col dottor Novarese, ricercatore del Dipartimento di Scienze Giuridiche ed Economiche dell'Università del Piemonte Orientale.

La nostra analisi si soffermerà solo su alcuni esperimenti svolti ad Alessandria e sui due effettuati presso la Facoltà di Economia di Torino. La fondamentale differenza tra le due categorie è riassunta nel seguente schema:

ALESSANDRIA → Umani

TORINO → Umani + agenti artificiali

Il primo esperimento ad Alessandria è stato realizzato presso la Facoltà di Giurisprudenza nel dicembre 2000 ed ha coinvolto 18 giocatori dislocati in più stanze.

I giocatori erano studenti reclutati nel corso di economia della stessa Facoltà ed hanno partecipato a quattro turni differenti, avendo a disposizione solo le informazioni strettamente legate agli aspetti tecnici del gioco (spiegazione di come interagire col terminale, delle regole del gioco, dell'impossibilità di dialogare con i compagni).

Data la scarsa adesione al gioco ed al conseguente ridotto numero di dati a disposizione per le analisi, si è deciso di considerare tali dati raccolti solo come parziali e da subito si è prospettata la necessità di approfondire lo studio con ulteriori esperimenti.

Il gioco *Somma 10*, presentato ai 18 giocatori, consiste nel coordinarsi individuando la combinazione giusta delle scelte, in modo tale da consentire ad ogni componente della squadra di ottenere il massimo punteggio possibile.

Ogni squadra è composta da tre giocatori, rimane invariata per tutta la durata dell'esperimento e l'identità dei suoi componenti è sconosciuta ai partecipanti. Ognuno deve scegliere un numero da giocare tra un set di valori messo a disposizione dal computer.

La scelta del numero è condizionata da due diversi ed opposti obiettivi:

- il primo è ottenere che la somma dei tre numeri scelti dal gruppo sia tale da raggiungere il più alto punteggio possibile secondo le regole dettate all'inizio del gioco;
- il secondo è "spendere" il meno possibile, visto che ogni numero ha un "costo" che è proporzionato al suo valore (il numero 0 ha costo 0,… , il numero 10 ha costo 10).

I punteggi decisi per l'esperimento sono i seguenti e sono uguali per tutti.

Se la somma dei tre numeri scelti dai membri della squadra è pari a 10, ciascuno vince una cifra pari a 40 meno il numero dichiarato.

Se la somma dei tre numeri è superiore a dieci, ciascuno vince una cifra pari a 30 meno il numero dichiarato.

Se la somma è inferiore a dieci non si vince nulla, ma si sopporta un costo pari al numero dichiarato.

Lo stesso gioco si ripete per 36 turni.

Sono stati effettuati quattro tipi di esperimenti diversi ognuno con l'intento di svelare un aspetto del comportamento del giocatore.

	insieme di numeri	**comunicazione numero turni**
Esperimento 1	0, 1, 3, 4, 10 per tutti i giocatori	si
Esperimento 2	0, 1, 3, 4, 10 per tutti i giocatori	no
Esperimento 3	a rotazione 0, 1, 2, 5, 10 0, 1, 5, 6, 10 0, 1, 3, 5, 10	si
Esperimento 4	a rotazione 0, 1, 2, 5, 10 0, 1, 5, 6, 10 0, 1, 3, 5, 10	no

Il primo esperimento prevedeva un set di numeri fissi (0, 1, 3, 4, 10) ed uguali per tutti i giocatori che erano stati messi a conoscenza del numero totale dei turni di gioco.

Il secondo prevedeva, diversamente dal precedente, che i giocatori non sapessero il numero dei turni di gioco, in modo da verificare se il comportamento sarebbe effettivamente cambiato, come sostiene la teoria dei giochi.

I primi due esperimenti utilizzavano un insieme di numeri fissi per aiutare i giocatori a raggiungere un accordo; negli altri due, invece, il set di numeri era diverso per ognuno in ciascun turno seguendo un sistema a rotazione.

All'inizio del gioco erano infatti messi a disposizione dal computer tre set di numeri ([0, 1, 2, 5, 10], [0, 1, 5, 6, 10] e [0, 1, 3, 5, 10]), uno per ogni giocatore della squadra; nei turni successivi la sequenza dei tre set ruotava in modo regolare; ogni quattro turni i numeri si ripetevano.

L'apprendimento organizzativo ed il coordinamento sono stati resi più difficili da raggiungere proprio per una maggiore varietà nelle possibilità di scelta.

Il terzo esperimento prevedeva di comunicare ai giocatori il numero dei turni mentre il quarto lo escludeva, sempre per verificare il variare delle scelte in funzione delle condizioni.

L'esperimento ha presentato qualche problema organizzativo. Il principale è stato la dislocazione dei giocatori nelle diverse aule, poiché se da un lato questa dispersione

era garanzia di sicurezza, precludendo ogni possibilità di scambio di informazioni tra i giocatori, dall'altro impediva il controllo da parte dei conduttori e la coordinazione essenziali per lo svolgimento del gioco.

L'analisi dei risultati ha fornito interessanti spunti di riflessione.

Tab. 1. Media della somma di numeri dichiarati per gruppo

1	11,0
2	10,6
3	11,3
4	10,2

La somma della terna è stata in media sempre superiore a 10. In pratica la paura di ottenere un punteggio negativo li ha spinti a giocare all'eccesso guadagnando un po' meno ma in maniera sicura e costante. I gruppi 2 e 4 hanno medie delle somme più vicine a 10 e sembrano più efficienti, mentre notiamo che le medie dei gruppi 1 e 3 sono decisamente più alte. In teoria ci saremmo aspettati che, conoscendo il numero dei turni, si verificasse un comportamento meno stabile, soprattutto verso la fine, come prevede la teoria dei giochi. Invece le strategie utilizzate dagli studenti sono state coerenti fino al termine della simulazione senza mostrare atteggiamenti egoistici nell'ultimo turno (che non sarebbero stati punibili dagli altri compagni-avversari), rispecchiando l'idea generale del bisogno di cooperazione necessaria per vincere.

Tab. 2. Media punti individuali ottenuti in ciascun turno per gruppo

Gruppo 1	26,0
Gruppo 2	28,0
Gruppo 3	24,3
Gruppo 4	32,3

Le medie dei punti individuali risultano essere intorno a 25 punti per turno; solo il quarto gruppo si discosta in maniera significativa con una media sorprendentemente superiore ai 30 punti.

La combinazione tra il non sapere il numero preciso dei turni e la presenza di un set di numeri variabile sembra essere ottimale rispetto agli altri casi. Questa situazione spinge maggiormente alla cooperazione e rende più elevate le medie dei punti anche se prevede un comportamento non del tutto efficiente.

Tab. 3. Distribuzione punti individuali ottenuti in ciascun turno per gruppo

Punti	1	2	3	4	Totale
-6	0	0	3	1	4
-5	0	0	24	10	34
-4	19	8	0	0	27
-3	33	59	19	8	119
-2	0	0	22	11	33
-1	6	11	33	14	64
0	53	15	10	4	82
20	74	40	33	7	154
24	0	0	21	2	23
25	0	0	157	45	202
26	74	62	0	0	136
27	35	50	51	14	150
28	0	0	40	4	44
29	6	9	15	7	37
30	57	12	11	2	82
35	0	0	72	173	245
36	81	126	0	0	207
37	162	252	66	173	653
38	0	0	66	173	239
40	48	4	5	0	57

Tab. 4. Varianza dei punteggi individuali ottenuti in ciascun turno per gruppo

1	194,4
2	182,1
3	176,0
4	112,5

A completamento dell'analisi sulla maggiore efficienza dei gruppi 2 e 4, notiamo che i punteggi, oltre ad avere medie più efficienti e più elevate, hanno minore varianza. In generale, sorprendentemente, c'è minore varianza proprio nei gruppi in cui sembrava più difficile coordinarsi.

Tab. 5. Distribuzione dei numeri dichiarati dai singoli giocatori nel corso del gioco per gruppo

Cifra	**1**	**2**	**3**	**4**	**Totale**
0	134	29	25	6	194
1	12	20	48	21	101
2	0	0	128	188	316
3	230	361	136	195	922
4	174	196	0	0	370
5	0	0	253	228	481
6	0	0	24	3	27
10	98	42	34	7	181

La selezione dei numeri è condizionata dalla scelta di una determinata strategia. Infatti troviamo nei primi turni (con il set fisso) con una maggiore frequenza lo 0, il 10 ed i numeri centrali 3 e 4, mentre le strategie con i set variabili sono più complesse ed articolate e lo comprova il fatto che i numeri hanno distribuzione più varia.

Tab. 6. Distribuzione delle somme delle terne in ciascun turno

	1	**2**	**3**	**4**
0	11	1		
1			1	
3	2	2		1
4	4	2	1	
5			2	2
6	2	4	7	2
7	4	6	2	1
8	7	4	15	6
9	7	12	9	4
10	105	128	70	173
11	25	22	19	4
12	7	1	21	4
13	8	2	36	11
14	3	8	6	3
15		2	8	2
16	2	8	1	
17	1	8	3	1
18	5	2	8	
20	18	2	4	2
21		1	2	
23	1		1	
24		1		
30	4			

Media per gruppo delle varianze delle scelte di ciascun giocatore

8,311	3,214	4,072	2,246

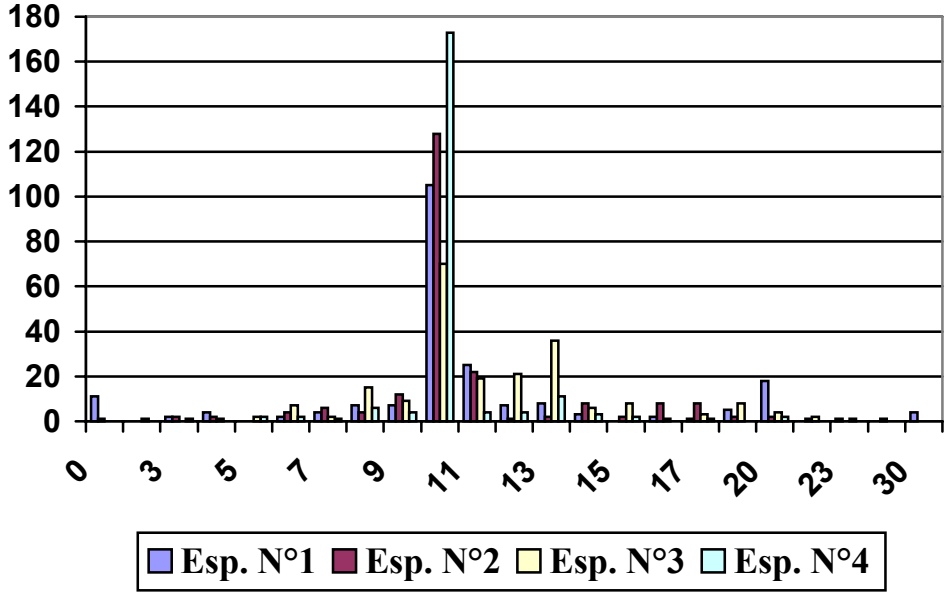
6.1.1 L'analisi generale dei dati

La lettura dei risultati può essere effettuata in due modi:

1) Confronto tra esperimenti in cui il set di numeri è fisso (1 e 2) ed esperimenti in cui il set di numeri è variabile (3 e 4)

A priori non era affatto scontato che i soggetti riuscissero a coordinarsi, dato che l'insieme dei numeri era variabile. A posteriori invece questo si è rivelato un buon modo per "guidare" i giocatori verso una strategia più cooperativa e meno egoistica.

Distribuzione delle somme



Negli esperimenti 3 e 4 i soggetti raggiungono risultati più efficienti ed equi. Infatti si riscontra un numero maggiore di casi in cui la somma risulta pari a 10, e un numero maggiore di casi in cui la somma è pari o superiore a 10, cioè di situazioni in cui si ottiene un *payoff* positivo.

Nei turni 3 e 4 la somma non risulta mai pari a 0 (infatti ci sono pochissimi casi in cui i giocatori dichiarano 0) la media dei punti è maggiore ed è minore la varianza.

2) Confronto tra esperimenti in cui i giocatori conoscevano il numero di turni (1 e 3) ed esperimenti in cui non lo conoscevano (3 e 4)

In questo caso le differenze sono ancora più nette di prima: i giocatori che non conoscono il numero di turni raggiungono risultati migliori.

La media delle somme raggiunte nei vari turni è più vicina a 10 per i gruppi 2 e 4. La media di punti è nettamente più alta e la varianza inferiore. Ci sono molti più turni in cui la cifra dichiarata è pari a 10 (o maggiore di 10).

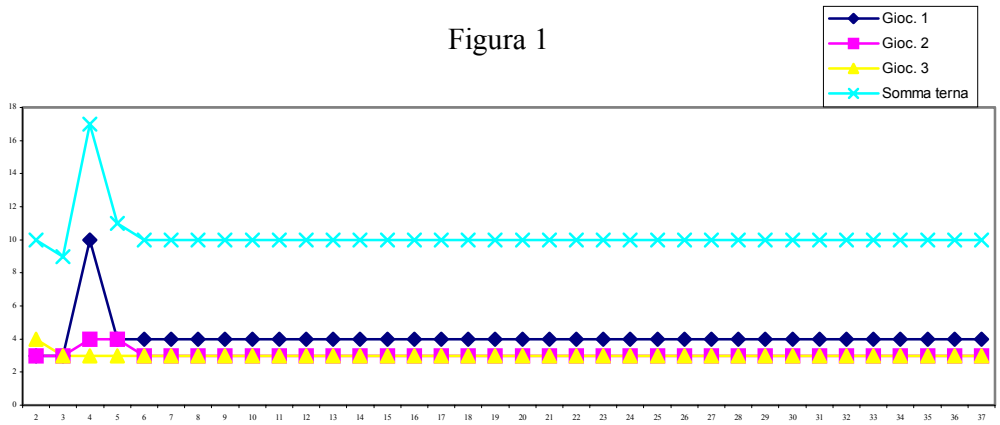
Negli esperimenti 1 e 3 non si registrano però variazioni particolarmente significative nei turni finali (la maggiore efficienza dei gruppi 3 e 4 si manifesta lungo tutto il gioco e sembra essere causata solo parzialmente da comportamenti strategici nei turni finali).

6.1.2 Le diverse strategie

Le strategie emergono in corrispondenza delle situazioni che si vanno a delineare nella terna.

Ogni terna sceglie la strategia che ritiene migliore o adotta quella su cui i tre si accordano prima.

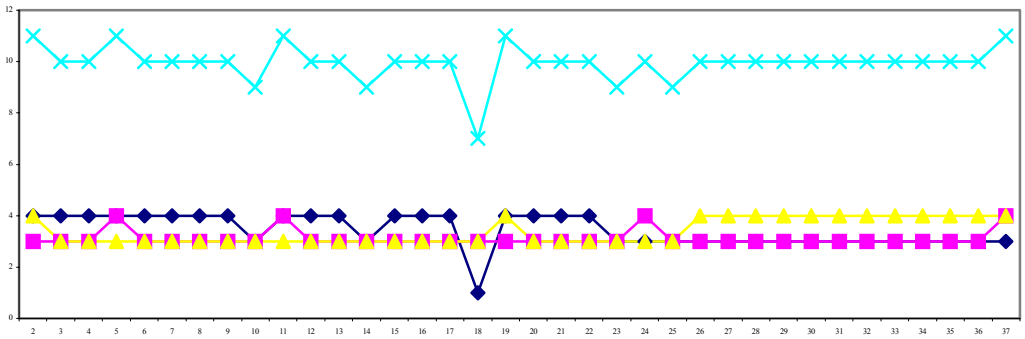
Figura 1



Gli equilibri stabili raggiunti dai terzetti negli esperimenti 1 e 3 prevedono che due giocatori dichiarino sempre 3 (per tutti i turni) e che un giocatore si sacrifichi dichiarando sempre 4 (come in figura 1). Non ci sono quindi equilibri stabili in cui i giocatori si alternano a dichiarare 3 e 4.

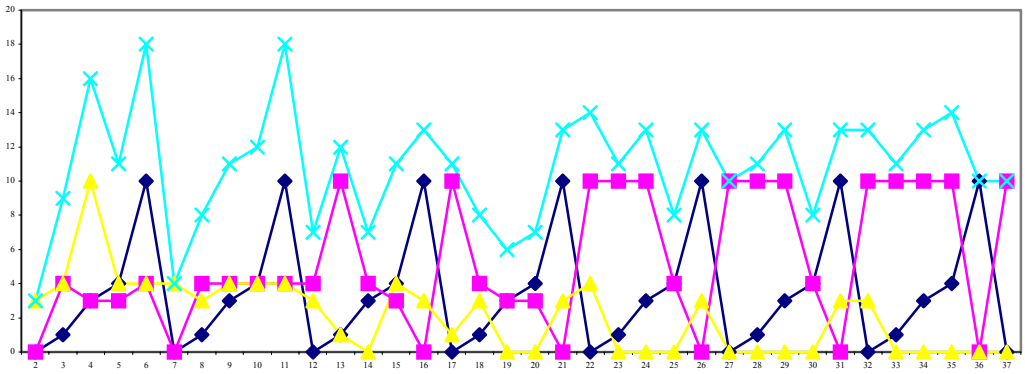
Ci sono stati casi in cui hanno provato ad alternarsi nel sacrificarsi, ma hanno portato a risultati inferiori per tutti (come in figura 2).

Figura 2



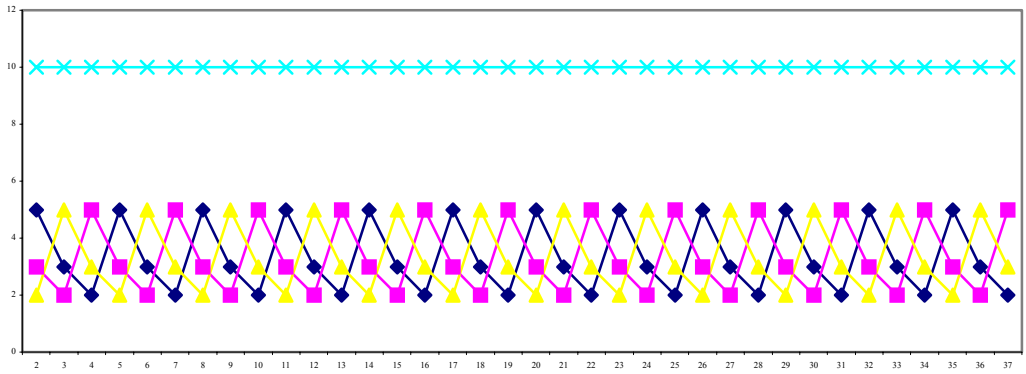
Non ci sono equilibri in cui i giocatori si alternano a dichiarare 10-0-0 (in linea generale non è più difficile da ottenere di un equilibrio 4-3-3, ma comporta evidentemente un maggiore rischio), ma solo sporadici tentativi falliti (come in figura 3).

Figura 3



Nel 3° e 4° gruppo i giocatori sono stati costretti a variare la propria scelta di turno in turno, così l'unico modo per ottenere un punteggio ottimale era quello di accordarsi e cooperare. L'idea di partenza era quella di costringere la squadra ad alternarsi nel sacrificio e molti hanno subito adottato la strategia più corretta. Trovare l'intesa restava comunque una cosa complicata, ma l'immediato accordo della terna rappresentata in figura 4 ha del miracoloso, i giocatori sono riusciti ad accordarsi dal immediatamente.

Figura 4



6.1.3 L'introduzione degli agenti artificiali

Questi giochi posso essere estesi e modificati con l'introduzione di algoritmi di programmazione in grado di simulare il comportamento umano, tali elementi sono denominati agenti artificiali. I vantaggi dell'introduzione di agenti non umani sono vari.

1) Si può pensare di sottoporre gruppi diversi di individui ad esperienze diverse (facendoli interagire con agenti artificiali diversi) e poi valutare gli effetti di tale "apprendimento" in una fase successiva del gioco.

2) Gli agenti artificiali possono anche svolgere un altro compito. Come ricordato, in questo esperimento entrano in gioco due aspetti: altruismo e ricerca di coordinamento. Se l'individuo sa di interagire con un agente artificiale (che utilizza una

data strategia) difficilmente sarà motivato dall'altruismo e quindi il suo comportamento sarà guidato dalla ricerca di un *payoff* ottimale.

Alcuni individui ad esempio accettano *payoff* costantemente più bassi degli altri. Tale scelta è motivata da forme di altruismo "sacrificarsi per il bene della squadra" oppure da forme di razionalità non troppo dissimili da quelle previste dalla teoria in relazione ad esempio all'*ultimatum game* (meglio guadagnare qualcosa che nulla), o addirittura dalla constatazione che non riuscendo a coordinarsi con gli altri membri della terna l'unica soluzione sia rimetterci qualcosa ma almeno ottenere un risultato di gruppo positivo.

Da una prima analisi si può constatare che con l'esperienza, e quindi con il proseguire dei turni, il punteggio migliora. Dopo un primo periodo di assestamento, per capire come funziona praticamente il gioco e per imparare ad applicare le regole da noi spiegate, si è potuto constatare che modificare il set di possibili scelte migliora leggermente la capacità di coordinamento.

6.2 Il primo esperimento di Torino

A Torino abbiamo effettuato un esperimento, modificato rispetto a quello di Alessandria, con un set di valori fissi (0,1,3,4,10) introducendo una fase di "apprendimento" con agenti artificiali. Il gioco mantiene le regole dell'esperimento analizzato nei paragrafi precedenti, ma è diviso in due parti: nella prima interagiscono agenti umani con giocatori artificiali guidati da un algoritmo, nella seconda i gruppi vengono ricomposti con i soli agenti umani.

La fase di apprendimento è stata inserita per verificare se sia possibile influenzare la strategia che il giocatore umano adotterà nella seconda fase del gioco.

Il comportamento che hanno tenuto gli agenti artificiali (altruisti o egoisti, forti o deboli) è stato studiato per permetterci di valutare se la fase di apprendimento:

- non aveva nessun effetto sul comportamento
- se dopo aver giocato con agenti egoisti si adottavano atteggiamenti altruisti
- se dopo aver giocato con agenti egoisti si preferiva un comportamento egoista.

Per poter scongiurare eventuali errori nella lettura dei risultati, abbiamo inserito un piccolo questionario prima dell'esperimento per testare la comprensione del gioco.

- Sapendo che gli altri due *partners* giocheranno rispettivamente 0 e 10, che numero dichiari?
- Sapendo che gli altri due *partners* giocheranno rispettivamente 3 e 4, che numero
dichiari?
- Sapendo che gli altri due *partners* giocheranno entrambi 0, che numero dichiari?
- Sapendo che gli altri due *partners* giocheranno 3, che numero dichiari?

Queste semplici domande sono state utili per valutare anche quanto tempo era necessario per poter comprendere a pieno il funzionamento del gioco.

Quando abbiamo pensato a come spiegare il gioco abbiamo prestato molta attenzione al significato di ogni parola utilizzata per poter essere il più chiari possibile nel descrivere le regole e le finalità.

Nonostante la particolare cura posta a questa fase dello svolgimento dell'esperimento, dall'analisi delle risposte date prima di cominciare sono venute alla luce numerose incomprensioni. Purtroppo infatti quando si spiegano le regole di un gioco si crea nella mente di chi ascolta una sorta di deformazione del gioco stesso: alcuni sono più attenti e disponibili e comprendono il giusto funzionamento subito, alcuni intuiscono male e ci impiegano parecchi turni a correggere la falsa idea che hanno del gioco, altri continuano fino alla fine a giocare un altro gioco. Inoltre sono serviti alcuni turni, almeno 6/8, per consentire ai partecipanti di interagire correttamente con l'*applet* che si presentava loro sul terminale: nonostante la semplice impostazione grafica della finestra di gioco, non è stato facile per alcuni imparare a giocare.

Innanzitutto era indispensabile appurare se nella fase di apprendimento i giocatori umani avevano seguito gli agenti artificiali adattandosi alla loro strategia per poter valutare le eventuali conseguenze.

Dai dati si può riscontrare una generale disponibilità all'adattamento del giocatore umano al gioco degli artificiali, anche se questo non si riflette quasi mai nel gioco tra umani. Il fatto che non sapessero della presenza di agenti artificiali, fa pensare che

adottare una strategia semplice o giocare un numero fisso aiuti a "scegliere" la propria strategia, altruista o egoista che sia.

6.2.1 Gli agenti artificiali

Ogni giocatore umano nella prima fase (15 turni) era affiancato da 2 agenti artificiali. La metà dei giocatori ha compiuto l'apprendimento con una coppia di agenti altruisti (uno forte e l'altro debole) e l'altra metà con agenti egoisti (sempre uno forte e l'altro debole).

Abbiamo inserito due agenti diversi tra loro per rendere meno prevedibile il loro comportamento e per permettere alla terna di raggiungere con più combinazioni 10 ottenendo il massimo punteggio.

Altruista debole:

```
        if (timer.cycles == 0) {
           b = 3;
         }
         else {
             if ((a.playerArray[a.id1].choice2 +
a.playerArray[a.id2].choice2) < 10) {
                if ((a.playerArray[a.id1].choice2 +
a.playerArray[a.id2].choice2 + 1) < 10){
                  if ((a.playerArray[a.id1].choice2 +
a.playerArray[a.id2].choice2 + 3) < 10){
                    if ((a.playerArray[a.id1].choice2 +
a.playerArray[a.id2].choice2 + 4) < 10){
                      b = 10;
                    }
                    else
                      b = 4;
                  }
                  else
                    b = 3;
                }
                else
                  b = 1;
             }
             else
                b = 0;
```

```
        }
          }
```

Al primo turno gioca 3, poi gioca adattandosi a ciò che succede nel turno precedente per arrivare almeno a 10. Se, per esempio, i suoi due compagni hanno giocato 0 e 4 il suo algoritmo lo guiderà a giocare 10.

Altruista forte: gioca sempre 4.

- *Egoista debole*:

```
   if (timer.cycles == 0)
    {
       b = 3;
    }
    else {
       if (((a.playerArray[a.id1].choice2 +
a.playerArray[a.id2].choice2) < 7 ||
(a.playerArray[a.id1].choice2 +
a.playerArray[a.id2].choice2) > 10))
        {
         b = 0;
        }
       else
         b = 3;
        }
    }
```

Al primo turno gioca 3, poi se la somma dei due compagni nel turno precedente è minore di 7 o maggiore di 10 gioca 0 (infatti se fosse minore 7 dovrebbe giocare troppo alto per un agente egoista, se fosse maggiore 10 cercherebbe di ridurre al minimo la spesa), altrimenti gioca 3.

Egoista forte: gioca sempre 3.

6.2.2 Modifiche al codice

Oltre all'inserimento degli agenti artificiali il programma ha subito numerose variazioni

1) Si era riscontrato un problema di perdita di concentrazione nei soggetti coinvolti nell'esperimento di Alessandria, che aveva rallentato notevolmente lo svolgimento allungando i tempi di esecuzione dei singoli turni. Infatti i soggetti dovevano effettuare manualmente l'aggiornamento dei dati dopo aver effettuato la scelta, ripetendo l'operazione più volte fino a quando tutti non avevano inviato la propria.

Abbiamo così modificato nel *Sum10Client* il sistema di trasmissione dei dati verso il *server*, ma soprattutto abbiamo automatizzato l'aggiornamento dei risultati al fine di ogni turno per dare il via libera ad una nuova scelta a tutti i giocatori contemporaneamente.

```
boolean check = false;
while (!check) {
long t = System.currentTimeMillis() + 5000;
textArea1.setText("Attendere.");
while (System.currentTimeMillis() < t) ;
if (!(askBlock2())) {
update();
check = true
```

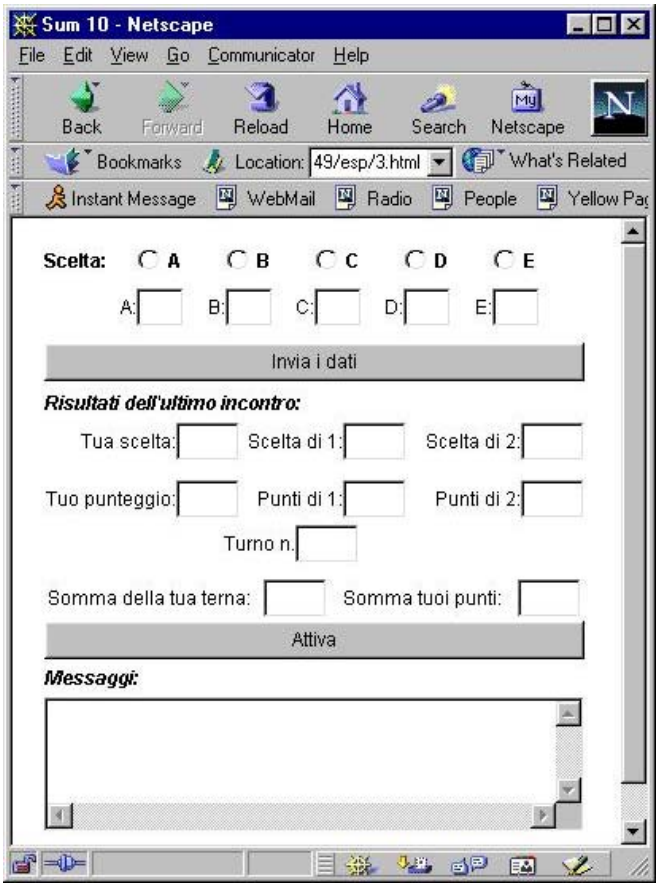
Una volta effettuata la scelta, utilizzando l'orologio interno del computer su cui si sta operando, ogni 5 secondi il *client* chiede il via libera al *server* per iniziare un nuovo turno. Solo quando il *server* ha raccolto le scelte di tutti aggiorna i risultati del gioco sullo schermo e permette di effettuare una nuova scelta. Questo automatismo è stato "sacrificato" per rendere l'*applet* più leggero possibile, ma ha dato buoni risultati nelle prove effettuate riducendo i tempi di attesa e di completamento dell'esperimento.

2) Un'altra modifica apportata al programma riguarda il *Player*, dove abbiamo affiancato ad agenti umani gli agenti artificiali. Gli uni giocano con gli altri contraddistinti solo dal numero identificativo (id) assegnato all'inizio della simulazione. Ogni agente segue due differenti passi: il primo permette all'agente guidato dall'umano di scegliere liberamente e all'agente virtuale di interpellare il *RuleMaster*, il secondo controlla che tutti, artificiali ed umani, abbiamo trasmesso la loro scelta.

3) La modifica più importante è stata l'introduzione del *RuleMaster* che "suggerisce" all'agente virtuale il comportamento da tenere secondo strategie di gioco precise formalizzate dagli algoritmi (studiati e progettati a tavolino per tale scopo) presentati sopra. Anche in questo caso esiste un solo *RuleMaster* per tutti gli agenti e solo il numero identificativo permette al singolo agente di seguire una precisa strategia.

4) Dopo numerosi tentativi abbiamo deciso di dotare l'*applet* di un pulsante per l'attivazione. Ogni *applet* viene inizializzato nel momento in cui viene richiamato; prima l'attivazione avveniva originariamente nel momento in cui l'*applet* era caricato dal *client* e rimaneva attivo anche nei tempi tecnici per la sistemazione in aule dei giocatori e per la spiegazione del gioco. Sui computer di Torino gli *applet* non reggevano il tempo necessario al completo svolgimento dell'esperimento: dopo averlo alleggerito al massimo, abbiamo deciso di renderlo attivo solo per il tempo strettamente necessario.

Potevamo fare caricare l'*applet* a ciascun giocatore al momento dell'avvio della simulazione, ma abbiamo preferito controllare noi l'apertura in modo che risultasse pronto per tutti nello steso momento. Con la presenza del pulsante per l'attivazione è stato possibile spiegare il gioco con l'*applet* visualizzato sullo schermo dei *client* davanti ai giocatori e attivarlo solo all'inizio del gioco.



Le informazioni a disposizione del giocatore comprendevano, oltre al punteggio totale e al numero del turno di gioco, anche i dati relativi al turno precedente, con le scelte della terna ed i relativi punteggi.

6.2.3 Le strategie individuate

In fase di programmazione avevamo previsto la nascita di strategie precise tra gli agenti umani, ma durante il gioco se ne sono spontaneamente create delle altre concettualmente diverse.

Avevamo ipotizzato due strategie possibili:

- una era di giocare un numero fisso durante tutto l'esperimento (giocatori egoisti ed altruisti);
- l'altra era di coordinarsi per giocare a turno il numero più elevato.

Dall'analisi dei risultati si possono invece individuare anche altre strategie oltre a quelle previste:

c'era chi giocava "con la squadra" per raggiungere il massimo punteggio possibile, anche sacrificandosi per più turni a giocare un numero più elevato;

c'era chi invece giocava "contro la squadra" cercando di spendere il meno possibile senza preoccuparsi di raggiungere 10.

All'interno delle strategie cooperative si possono individuare due diverse terne di numeri giocate durante l'esperimento: 4,3,3 e 10,0,0. L'alternanza nel giocare il numero più alto non si è verificata in maniera sistematica, anche se è stata tentata, per la difficoltà nel passare da un equilibrio (4,3,3…) all'altro (3,4,3…). Hanno ottenuto punteggi più elevati infatti le terne in cui uno dei giocatori ha accettato di "sacrificarsi" per il bene (suo e) della squadra spendendo più degli altri.

Chi ha giocato "contro la squadra" ha sistematicamente giocato 0 senza dare nessun apporto personale al punteggio ottenuto dalla terna. Non può essere considerato un atteggiamento da egoista puro, come da noi ipotizzato, perché non è riscontrabile il tentativo di massimizzare il profitto minimizzando i costi, ma semplicemente di spendere il meno possibile.

6.2.4 I preparativi

Il progetto che ha ispirato il nostro esperimento, come già scritto, è partito dalla Facoltà di Giurisprudenza di Alessandria dove sono stati effettuati i primi esperimenti svolti con l'aiuto delle simulazioni l'anno scorso.

Abbiamo collaborato con il dott. Novarese del Dipartimento di scienze giuridiche ed economiche dell'Università del Piemonte Orientale fin dall'inizio e questo ci ha permesso di acquisire la loro esperienza nel campo degli esperimenti evitando di partire da zero ed incappare in errori di inesperienza già riscontrati precedentemente.

Per effettuare l'esperimento è stato utilizzato Swarm come piattaforma informatica, ed in particolare il programma Swiee sviluppato dal dott. Boero, laureatosi presso l'Università di Economia di Torino.

Swiee si basa sulla combinazione tra la solidità e la praticità di Swarm e la versatilità del linguaggio di programmazione Java, risultando uno strumento perfetto

come base per la programmazione di ogni esperimento economico. Per svolgere l'esperimento presso la sede della Facoltà di economia di Torino abbiamo usufruito dell'aiuto del dipartimento di scienze economiche e finanziarie "Giuseppe Prato" per organizzare l'esperimento e delle aule del Liases per effettuarlo.

Avendo scelto di ricercare comportamenti altruistici e cooperativi attraverso una variante del gioco *Somma 10,* siamo stati indecisi se coinvolgere studenti del primo anno con poca esperienza di teorie economiche o studenti più "esperti" del terzo anno.

Alla fine si è deciso di coinvolgere entrambe le classi di studenti. In questo modo abbiamo utilizzato un campione più omogeneo ed è stato possibile analizzare i risultati ottenuti anche in base all'esperienza del singolo giocatore, per valutare se questa influenzasse o meno le scelte all'interno del gioco. Inoltre abbiamo potuto coinvolgere più giocatori grazie alla "promozione" fattaci da alcuni professori del dipartimento tra gli studenti dei loro corsi di economia.

All'inizio abbiamo chiesto la collaborazione dei docenti più vicini all'esperimento:

-il professor Bagliano per Economia monetaria del terzo/quarto anno

-il professor Frigero per Economia politica 1 del primo anno

-il professor Sembenelli per Economia applicata e Econometria del terzo/quarto anno

-il Professor Terna per Tecniche di previsione economica (reti neurali) e Dinamica economica del terzo e quarto anno.

Poi abbiamo allargato l'invito ad altri corsi avendo riscontrato che le adesioni erano scarse e procedevano a rilento:

-la professoressa Fornero per Economia politica 2 del secondo anno

-il professor De Battistini per Economia politica 2 del secondo anno

-il professor Bortolotti per Economia delle istituzioni (*no profit*) del terzo/quarto anno

-il professor Siniscalco per Economia politica del primo anno

L'annuncio del gioco e la spiegazione delle condizioni per la partecipazione è stata fatta agli studenti ben tre mesi prima dell'esperimento. A tutti i giocatori è stato spiegato in aula il regolamento.

Il contenuto del "comunicato" è stato preparato, studiato e discusso in una riunione con tutti gli organizzatori, avendo cura di non svelare il funzionamento del

gioco (che è stato descritto solo prima dell'esperimento), ma di incuriosire gli studenti ed invogliarli a partecipare senza creare in loro false illusioni.

Il testo utilizzato per il reclutamento è il seguente:

Esperimenti economici al Liases

Giovedì 3 maggio 2001, dalle ore 18 in poi, si terrà presso le aule del Liases della Facoltà di Economia in corso Unione Sovietica 218 bis, un esperimento economico collegato a due tesi di laurea.

La sperimentazione consisterà in un gioco a punteggio, da effettuare su computer, della durata di circa un'ora.

I migliori classificati riceveranno premi in buoni libri (proporzionati ai risultati ottenuti, per un massimo di 30.000 lire a testa) da spendere presso la libreria Celid all'interno della Facoltà.

Non sono richieste conoscenze specifiche.

Chi fosse interessato è pregato di dare la propria adesione entro il 31 marzo.

Verranno in seguito pubblicati gli elenchi con l'indicazione precisa dell'ora e dell'aula, affiggendoli presso le bacheche del Dipartimento di scienze economiche e finanziarie G. Prato e del Palazzo del lavoro.

Per chiarimenti rivolgersi al docente relatore delle tesi in questione, Pietro Terna pietro.terna@unito.it, o agli studenti che svolgono le tesi: Andrea Gianoglio andrea.gianoglio@tin.it e Filippo Pavesio f.pavesio@tin.it.

Allegato al comunicato c'era un foglio di adesione sul quale abbiamo potuto raccogliere i primi dati sugli studenti disposti a svolgere l'esperimento.

Foglio di adesione ***(scrivere in stampatello)***

Nome	*Cognome*	*Matricola*	*Indirizzo e-mail (preferito) o recapito telefonico*

Fortunatamente la maggior parte degli studenti che ha aderito all'esperimento ha indicato come recapito un indirizzo e-mail; in questo modo è stato possibile anche mandare diversi avvisi per sollecitare la loro partecipazione, come previsto in fase di progettazione.

Il primo avviso è stato spedito una settimana prima della simulazione volendo ricordare loro solamente l'impegno preso; insieme alle e-mail abbiamo dovuto spedire anche numerosi *short message* per contattare coloro i quali avevano indicato solo il numero di cellulare.

Purtroppo il numero dei partecipanti stimato dagli elenchi è diminuito perché alcuni hanno risposto al nostro messaggio scusandosi per non poter partecipare e alcuni indirizzi sono tornati al mittente risultando inesistenti. Infatti molti indirizzi, scritti in corsivo, erano poco leggibili e in fase di riscrittura, notando questa difficoltà, avevamo ipotizzato che alcuni potessero poi risultare non corretti.

Il secondo avviso è stato spedito il giorno prima della data fissata per l'esperimento con l'indicazione precisa dell'ora per ciascun gruppo. Abbiamo dato per scontato che tutti leggessero la posta quotidianamente, ma forse analizzando la cosa a posteriori avremmo dovuto anticipare almeno di un giorno l'ultimo avviso. Infatti abbiamo raccolto le lamentele di alcuni che, pur essendo venuti all'ora indicata nel primo messaggio, sostenevano di non aver ricevuto il secondo, non avendo letto la posta il giorno prima.

Abbiamo dovuto riflettere anche sulla motivazione da dare ai giocatori per rendere credibile il risultato dell'esperimento. L'esperienza di Alessandria anche in questo caso ci è stata molto utile come termine di paragone; infatti in quel caso l'esperimento è stato inserito come seminario all'interno di un corso di economia della facoltà di giurisprudenza. In questo modo la partecipazione all'esperimento è diventa parte integrante del programma del corso ed il suo risultato ha influenzato in maniera lieve, ma allettante il voto finale.

Per il nostro esperimento abbiamo comunque deciso di muoverci in maniera differente, pur cercando di motivare alla partecipazione.

Eravamo infatti indecisi sullo strumento da utilizzare quando, fortunatamente, ci sono venuti in aiuto il codice deontologico dell'economista sperimentale e le indicazioni

della letteratura in materia, che giudicano la leva economica come la migliore. Utilizzando premi in denaro si rende la ricerca e l'esperimento credibile anche al di fuori del contesto universitario locale. Per evitare di dover pagare i premi in contanti abbiamo pensato di mettere in palio buoni libri da ritirare presso la libreria della Facoltà.

La scelta del valore da assegnare alle singole ricompense ci ha posto di fronte ad un nuovo bivio. Non sapevamo se dividere i vincitori in classi di merito in base al punteggio finale ottenuto e pagare cifre prestabilite o ripagare le vincite in maniera proporzionale. Poiché il punteggio finale dipendeva non solo dall'abilità del singolo, ma soprattutto dalla capacità di coordinamento del gruppo appresa nei turni di training e la graduatoria era personale, non volevamo svantaggiare troppo coloro che giocavano con compagni meno motivati o meno disposti a collaborare e così alla fine abbiamo deciso di utilizzare le classi di merito.

Scelto il metodo ci restava solo da decidere il valore dei singoli premi in maniera da invogliare i giocatori al massimo impegno e ricompensare il maggior numero di persone possibile.

Avevamo ripartito i fondi a nostra disposizione in questo modo:

8 premi da £30.000

8 premi da £15.000

14 premi da £10.000.

Visto poi che nelle due aule si sarebbero svolti due esperimenti leggermente diversi e che le informazioni date ai due gruppi di giocatori sarebbero state diverse, non ci sembrava giusto compilare una sola graduatoria per le due aule.

Infatti la differenza nelle informazioni possedute o nelle condizioni in cui si effettuava la fase di apprendimento poteva condizionare lo svolgimento ed il risultato finale del gioco. Per questo abbiamo deciso di stilare due graduatorie diverse, una per aula, e premiare i migliori delle due classifiche (4 da £30.000, 4 da £15.000 e 7 da £10.000 per aula).

Scelto il campione abbiamo dovuto organizzare l'esperimento vero e proprio, con tutti i problemi tecnici legati alla realizzazione simultanea del maggior numero possibile di casi.

Grazie alla disponibilità del prof. Margarita, responsabile del Liases, non solo abbiamo avuto a disposizione le sale informatiche dove svolgere fisicamente l'esperimento con più di cinquanta computer in rete, ma abbiamo usufruito anche del supporto pratico dei tecnici, utile per la risoluzione dei non pochi problemi informatici sorti durante la preparazione dello stesso.

Le aule che hanno ospitato l'esperimento contengono ognuna venticinque computer numerati ed identificabili su quattro file e sono collegati in una rete locale condivisa da tutte le macchine della facoltà.

I *server* utilizzati erano un Pentium 200 Mhz con 64 MB di Ram e un Pentiun III 650 Mhz con 160 Mb di Ram, mentre i computer su cui "giocavano" gli studenti erano Pentium 100.

All'inizio pensavamo di dividere l'esperimento in due sessioni da svolgersi in due giorni diversi per cercare di coinvolgere più giocatori possibili.

Poi abbiamo deciso di fare tutto in un giorno solo con due turni consecutivi. Abbiamo convocato i giocatori del secondo turno circa un'ora dopo rispetto a quelli del primo turno e li abbiamo trattenuti in una stanza adiacente a quella dell'esperimento. In questo modo quelli di un turno completavano il gioco e lasciavano l'aula senza incontrare i giocatori del successivo impedendo così il passaggio di informazioni. Temevamo infatti che il contatto tra i "turni" potesse alterare i risultati condizionando il comportamento nel gioco.

6.2.5 Alcuni problemi riscontrati

Nonostante si sia cercato, in fase di studio dell'esperimento, di immaginare tutti i possibili intoppi a cui si sarebbe andati incontro il giorno della simulazione, si sono puntualmente verificati imprevisti, mentre altre difficoltà che invece eravamo quasi certi di riscontrare non si sono presentate.

Si presumeva, infatti, di avere problemi di partecipazione, non essendo affatto sicuri della coerenza di coloro che si erano iscritti e che erano stati più volte contattati da noi. L'affluenza, invece, è stata ottima e qualitativamente molto convincente: la maggior parte degli studenti sembrava motivata e impaziente di giocare.

La leva economica è sicuramente servita come stimolo, ma molti studenti erano curiosi e interessati all'esperimento in sé, molti alla fine dei turni previsti, hanno chiesto spiegazioni sul suo funzionamento, sui codici del programma e sulle finalità della ricerca.

Sono arrivati all'ora prevista praticamente tutti quelli convocati per il primo turno e sono stati fatti accomodare nell'aula di fronte al Liases.

Nell'assegnazione dei posti abbiamo fatto particolare attenzione a non mettere vicini i componenti di una stessa squadra, sparpagliandoli per le aule, per impedire che si influenzassero.

Il primo turno era composto da 18 persone, 9 per aula.

Hanno compiuto senza intoppi due turni interi, poi il programma si è bloccato impedendo il proseguimento; abbiamo tentato di riavviare la simulazione, ma non è più stato possibile ripristinarla efficacemente. È stato deciso allora di effettuare l'esperimento a gruppi di sei per non appesantire il programma e poterne controllare meglio lo svolgimento.

Purtroppo i primi 18 concorrenti erano a quel punto "compromessi" e non hanno potuto partecipare nuovamente all'esperimento. Non sembrandoci giusto escluderli dalle graduatorie solo per un intoppo nell'esecuzione del programma, abbiamo deciso di dar loro un premio di partecipazione di £10000, dovendo così ristudiare la ripartizione delle vincite.

Abbiamo assegnato ai due gruppi 3 premi da £30000 e 4 da £15000 eliminando quelli da £10000 (in pratica già destinati alle prime "cavie").

Dovendo ristabilire i turni il nostro timore era che la maggior parte dei concorrenti non fosse disposta ad aspettare fino a tardi per completare il proprio. Invece data loro la "cattiva notizia" hanno fatto la fila per iscriversi nei cinque turni da noi previsti. Se avessimo avuto maggiore tempo saremmo riusciti a far giocare tutti; purtroppo molti hanno dovuto rinunciare. Alcuni giocatori hanno aspettato fino alle 21 senza mostrare alcun segno di insofferenza, anzi apparendo divertiti e partecipando attivamente al gioco.

Il problema principale che ha condizionato pesantemente lo svolgimento dell'esperienza è stato rappresentato dalle macchine utilizzate[23]. Infatti, come già spiegato, il programma opera su un *server* e le "stazioni di gioco" si collegano a questo interagendo per mezzo dell'*applet*; le comunicazioni tra i vari computer avvengono tramite la rete locale della facoltà. La lentezza delle "stazioni" ha condizionato il passaggio di informazioni a tal punto da bloccare il *browser* Netscape ed impedire il corretto svolgimento dell'esperimento. Dopo aver registrato un certo numero di piccoli o grandi intoppi, siamo riusciti a completare l'esperimento con 27 persone. I primi due gruppi hanno effettuato i turni di training con agenti virtuali egoisti, mentre i seguenti due con agenti altruisti. L'ultimo gruppo è stato messo a conoscenza del fatto che durante il periodo di training avrebbero giocato con agenti virtuali, mentre agli altri è stato tenuto nascosto.

6.2.6 I dati raccolti

I dati raccolti durante l'esperimento sono stati trascritti in diversi file a seconda del turno di gioco e del *server* che operava. I due *server* sono stati denominati con le lettere (*a* e *b*) e i turni con i numeri (da 1 a 4 per *a* e da 1 a 5 per il *b*).

Da tali file sono state tratte le seguenti tabelle riassuntive. Sono state elencate le scelte dei soli agenti umani, i primi quindici turni quindi riportano dati riferiti a tre terne diverse.

LEGENDA

OBS: indica il numero dell'osservazione.

Turno: indica il numero del turno a cui si riferiscono i dati della riga.

Somma: indica la somma della terna (nei primi quindici turni non è indicato il valore perché le tre scelte sono riferite alla fase di apprendimento con gli agenti artificiali).

[23] Anche se come vedremo analizzando il secondo esperimento svolto a Torino il 19 aprile del 2002 il problema reale non sta nella "qualità" dei client, bensì nell'architettura del software sottostante (per approfondimenti vedi par.)

Scelta1\2\3: indica la scelta del giocatore1\2\3.

Appr1\2\3: indica il tipo di apprendimento effettuato dal giocatore 1\2\3 (afad = altruista forte e debole) (edef = egoista debole e forte).

GIOCO=arun1

OBS	TURNO	SOMMA	SCELTA 1	SCELTA 2	SCELTA 3	APPR 1	APPR 2	APPR 3
1	1		4	0	0	afad	edef	afad
2	2		3	4	1	afad	edef	afad
3	3		4	4	0	afad	edef	afad
4	4		3	4	0	afad	edef	afad
5	5		3	4	0	afad	edef	afad
6	6		3	4	1	afad	edef	afad
7	7		3	4	3	afad	edef	afad
8	8		3	4	0	afad	edef	afad
9	9		3	4	3	afad	edef	afad
10	10		3	4	3	afad	edef	afad
11	11		3	4	3	afad	edef	afad
12	12		3	4	3	afad	edef	afad
13	13		3	4	0	afad	edef	afad
14	14		3	4	3	afad	edef	afad
15	15		3	4	0	afad	edef	afad
16	16	6	3	3	0	afad	edef	afad
17	17	7	3	3	1	afad	edef	afad
18	18	6	3	3	0	afad	edef	afad
19	19	16	10	3	3	afad	edef	afad
20	20	6	3	3	0	afad	edef	afad
21	21	9	3	3	3	afad	edef	afad
22	22	10	3	3	4	afad	edef	afad
23	23	6	3	3	0	afad	edef	afad
24	24	10	3	3	4	afad	edef	afad
25	25	10	3	3	4	afad	edef	afad
26	26	10	3	3	4	afad	edef	afad
27	27	6	3	3	0	afad	edef	afad
28	28	10	3	3	4	afad	edef	afad
29	29	10	3	3	4	afad	edef	afad
30	30	6	3	3	0	afad	edef	afad

GIOCO=arun2

OBS	TURNO	SOMMA	SCELTA 1	SCELTA 2	SCELTA 3	APPR 1	APPR 2	APPR 3
31	1		4	4	4	afad	edef	afad
32	2		4	3	3	afad	edef	afad
33	3		4	3	4	afad	edef	afad
34	4		4	10	10	afad	edef	afad
35	5		3	3	3	afad	edef	afad
36	6		3	10	4	afad	edef	afad
37	7		3	0	4	afad	edef	afad
38	8		3	10	4	afad	edef	afad
39	9		3	0	4	afad	edef	afad
40	10		3	0	4	afad	edef	afad
41	11		3	10	4	afad	edef	afad
42	12		3	0	4	afad	edef	afad
43	13		3	10	4	afad	edef	afad
44	14		3	10	4	afad	edef	afad
45	15		3	0	4	afad	edef	afad
46	16	9	3	3	3	afad	edef	afad
47	17	10	3	3	4	afad	edef	afad
48	18	10	3	3	4	afad	edef	afad
49	19	10	3	3	4	afad	edef	afad
50	20	6	3	3	0	afad	edef	afad
51	21	4	3	0	1	afad	edef	afad
52	22	7	0	4	3	afad	edef	afad
53	23	6	3	3	0	afad	edef	afad
54	24	4	0	4	0	afad	edef	afad
55	25	3	0	3	0	afad	edef	afad
56	26	0	0	0	0	afad	edef	afad
57	27	1	0	0	1	afad	edef	afad
58	28	11	0	10	1	afad	edef	afad
59	29	3	0	0	3	afad	edef	afad
60	30	4	0	4	0	afad	edef	afad

GIOCO=arun3

OBS	TURNO	SOMMA	SCELTA 1	SCELTA 2	SCELTA 3	APPR 1	APPR 2	APPR 3
61	1		4	3	10	edef	afad	edef
62	2		4	4	4	edef	afad	edef
63	3		4	3	10	edef	afad	edef
64	4		4	3	10	edef	afad	edef
65	5		4	3	3	edef	afad	edef
66	6		4	3	10	edef	afad	edef
67	7		4	3	10	edef	afad	edef
68	8		4	3	10	edef	afad	edef
69	9		4	3	0	edef	afad	edef
70	10		4	3	10	edef	afad	edef
71	11		4	3	10	edef	afad	edef
72	12		4	3	10	edef	afad	edef
73	13		4	3	10	edef	afad	edef
74	14		4	3	1	edef	afad	edef
75	15		4	3	1	edef	afad	edef
76	16	16	3	3	10	edef	afad	edef
77	17	10	0	0	10	edef	afad	edef
78	18	10	0	0	10	edef	afad	edef
79	19	10	0	0	10	edef	afad	edef
80	20	10	0	0	10	edef	afad	edef
81	21	10	0	0	10	edef	afad	edef
82	22	10	0	0	10	edef	afad	edef
83	23	10	0	0	10	edef	afad	edef
84	24	10	0	0	10	edef	afad	edef
85	25	0	0	0	0	edef	afad	edef
86	26	21	10	10	1	edef	afad	edef
87	27	1	0	0	1	edef	afad	edef
88	28	3	0	0	3	edef	afad	edef
89	29	24	4	10	10	edef	afad	edef
90	30	5	4	0	1	edef	afad	edef

GIOCO=arun4

OBS	TURNO	SOMMA	SCELTA 1	SCELTA 2	SCELTA 3	APPR 1	APPR 2	APPR 3
91	1		3	3	10	edef	afad	edef
92	2		10	4	3	edef	afad	edef
93	3		10	3	10	edef	afad	edef
94	4		10	3	10	edef	afad	edef
95	5		4	3	10	edef	afad	edef
96	6		10	3	3	edef	afad	edef
97	7		10	3	0	edef	afad	edef
98	8		10	3	0	edef	afad	edef
99	9		10	3	0	edef	afad	edef
100	10		10	3	0	edef	afad	edef
101	11		10	3	0	edef	afad	edef
102	12		10	3	0	edef	afad	edef
103	13		10	3	0	edef	afad	edef
104	14		4	3	0	edef	afad	edef
105	15		10	3	0	edef	afad	edef
106	16	8	4	4	0	edef	afad	edef
107	17	9	3	3	3	edef	afad	edef
108	18	18	4	4	10	edef	afad	edef
109	19	9	3	3	3	edef	afad	edef
110	20	11	4	3	4	edef	afad	edef
111	21	9	3	3	3	edef	afad	edef
112	22	11	4	3	4	edef	afad	edef
113	23	11	3	4	4	edef	afad	edef
114	24	9	1	4	4	edef	afad	edef
115	25	17	3	4	10	edef	afad	edef
116	26	9	1	4	4	edef	afad	edef
117	27	11	4	4	3	edef	afad	edef
118	28	8	0	4	4	edef	afad	edef
119	29	17	10	4	3	edef	afad	edef
120	30	18	4	4	10	edef	afad	edef

GIOCO=brun1

OBS	TURNO	SOMMA	SCELTA 1	SCELTA 2	SCELTA 3	APPR 1	APPR 2	APPR 3
121	1		10	3	4	afad	edef	afad
122	2		3	3	3	afad	edef	afad
123	3		3	10	3	afad	edef	afad
124	4		3	0	3	afad	edef	afad
125	5		4	4	3	afad	edef	afad
126	6		4	0	3	afad	edef	afad
127	7		10	4	3	afad	edef	afad
128	8		1	3	3	afad	edef	afad
129	9		3	4	3	afad	edef	afad
130	10		4	4	3	afad	edef	afad
131	11		4	4	3	afad	edef	afad
132	12		10	4	3	afad	edef	afad
133	13		3	4	3	afad	edef	afad
134	14		4	4	3	afad	edef	afad
135	15		3	4	3	afad	edef	afad
136	16	4	1	0	3	afad	edef	afad
137	17	13	3	0	10	afad	edef	afad
138	18	3	3	0	0	afad	edef	afad
139	19	6	3	0	3	afad	edef	afad
140	20	8	4	4	0	afad	edef	afad
141	21	11	4	4	3	afad	edef	afad
142	22	10	4	3	3	afad	edef	afad
143	23	10	4	3	3	afad	edef	afad
144	24	10	4	3	3	afad	edef	afad
145	25	10	4	3	3	afad	edef	afad
146	26	7	4	3	0	afad	edef	afad
147	27	16	10	3	3	afad	edef	afad
148	28	10	4	3	3	afad	edef	afad
149	29	10	4	3	3	afad	edef	afad
150	30	10	4	3	3	afad	edef	afad

GIOCO=brun2

OBS	TURNO	SOMMA	SCELTA 1	SCELTA 2	SCELTA 3	APPR 1	APPR 2	APPR 3
151	1		3	4	4	afad	edef	afad
152	2		3	3	3	afad	edef	afad
153	3		3	3	3	afad	edef	afad
154	4		3	4	4	afad	edef	afad
155	5		3	3	3	afad	edef	afad
156	6		3	3	3	afad	edef	afad
157	7		3	4	3	afad	edef	afad
158	8		3	3	1	afad	edef	afad
159	9		3	3	3	afad	edef	afad
160	10		3	4	4	afad	edef	afad
161	11		3	10	3	afad	edef	afad
162	12		3	4	3	afad	edef	afad
163	13		3	4	3	afad	edef	afad
164	14		3	3	3	afad	edef	afad
165	15		3	3	3	afad	edef	afad
166	16	10	3	4	3	afad	edef	afad
167	17	9	3	3	3	afad	edef	afad
168	18	11	3	4	4	afad	edef	afad
169	19	9	3	3	3	afad	edef	afad
170	20	9	3	3	3	afad	edef	afad
171	21	11	3	4	4	afad	edef	afad
172	22	11	3	4	4	afad	edef	afad
173	23	9	3	3	3	afad	edef	afad
174	24	10	3	3	4	afad	edef	afad
175	25	10	3	3	4	afad	edef	afad
176	26	10	3	3	4	afad	edef	afad
177	27	10	3	3	4	afad	edef	afad
178	28	10	3	3	4	afad	edef	afad
179	29	10	3	3	4	afad	edef	afad
180	30	10	3	3	4	afad	edef	afad

GIOCO=brun3

OBS	TURNO	SOMMA	SCELTA 1	SCELTA 2	SCELTA 3	APPR 1	APPR 2	APPR 3
181	1		4	3	3	edef	afad	edef
182	2		3	0	0	edef	afad	edef
183	3		3	1	0	edef	afad	edef
184	4		4	10	3	edef	afad	edef
185	5		10	3	0	edef	afad	edef
186	6		10	0	3	edef	afad	edef
187	7		10	0	3	edef	afad	edef
188	8		10	0	3	edef	afad	edef
189	9		4	0	0	edef	afad	edef
190	10		10	0	3	edef	afad	edef
191	11		4	0	3	edef	afad	edef
192	12		10	0	3	edef	afad	edef
193	13		4	0	3	edef	afad	edef
194	14		0	0	3	edef	afad	edef
195	15		0	0	3	edef	afad	edef
196	16	10	10	0	0	edef	afad	edef
197	17	4	4	0	0	edef	afad	edef
198	18	10	10	0	0	edef	afad	edef
199	19	4	4	0	0	edef	afad	edef
200	20	10	10	0	0	edef	afad	edef
201	21	7	4	3	0	edef	afad	edef
202	22	16	10	3	3	edef	afad	edef
203	23	10	4	3	3	edef	afad	edef
204	24	6	0	3	3	edef	afad	edef
205	25	3	0	3	0	edef	afad	edef
206	26	10	10	0	0	edef	afad	edef
207	27	4	4	0	0	edef	afad	edef
208	28	6	0	3	3	edef	afad	edef
209	29	0	0	0	0	edef	afad	edef
210	30	7	4	0	3	edef	afad	edef

GIOCO=brun4

OBS	TURNO	SOMMA	SCELTA 1	SCELTA 2	SCELTA 3	APPR 1	APPR 2	APPR 3
211	1		4	3	4	edef	afad	edef
212	2		4	1	3	edef	afad	edef
213	3		4	3	4	edef	afad	edef
214	4		4	4	3	edef	afad	edef
215	5		4	1	3	edef	afad	edef
216	6		4	1	3	edef	afad	edef
217	7		4	3	4	edef	afad	edef
218	8		4	1	3	edef	afad	edef
219	9		4	1	3	edef	afad	edef
220	10		4	3	4	edef	afad	edef
221	11		4	0	3	edef	afad	edef
222	12		4	3	3	edef	afad	edef
223	13		4	3	3	edef	afad	edef
224	14		4	3	3	edef	afad	edef
225	15		4	3	3	edef	afad	edef
226	16	9	3	3	3	edef	afad	edef
227	17	8	3	1	4	edef	afad	edef
228	18	11	3	4	4	edef	afad	edef
229	19	9	3	3	3	edef	afad	edef
230	20	7	3	1	3	edef	afad	edef
231	21	8	3	1	4	edef	afad	edef
232	22	16	3	3	10	edef	afad	edef
233	23	11	3	4	4	edef	afad	edef
234	24	10	3	3	4	edef	afad	edef
235	25	10	3	3	4	edef	afad	edef
236	26	9	3	3	3	edef	afad	edef
237	27	10	3	4	3	edef	afad	edef
238	28	11	3	4	4	edef	afad	edef
239	29	9	3	3	3	edef	afad	edef
240	30	9	3	3	3	edef	afad	edef

GIOCO=brun5

OBS	TURNO	SOMMA	SCELTA 1	SCELTA 2	SCELTA 3	APPR 1	APPR 2	APPR 3
241	1		1	3	4	edef	afad	edef
242	2		3	4	3	edef	afad	edef
243	3		1	10	3	edef	afad	edef
244	4		1	1	3	edef	afad	edef
245	5		0	4	3	edef	afad	edef
246	6		0	3	3	edef	afad	edef
247	7		0	1	3	edef	afad	edef
248	8		10	3	3	edef	afad	edef
249	9		3	0	3	edef	afad	edef
250	10		3	3	3	edef	afad	edef
251	11		0	0	3	edef	afad	edef
252	12		0	3	3	edef	afad	edef
253	13		0	0	3	edef	afad	edef
254	14		0	3	3	edef	afad	edef
255	15		0	0	3	edef	afad	edef
256	16	13	0	10	3	edef	afad	edef
257	17	13	0	10	3	edef	afad	edef
258	18	10	0	10	0	edef	afad	edef
259	19	0	0	0	0	edef	afad	edef
260	20	13	0	10	3	edef	afad	edef
261	21	5	0	4	1	edef	afad	edef
262	22	13	0	10	3	edef	afad	edef
263	23	2	0	1	1	edef	afad	edef
264	24	0	0	0	0	edef	afad	edef
265	25	13	0	10	3	edef	afad	edef
266	26	3	0	0	3	edef	afad	edef
267	27	7	0	4	3	edef	afad	edef
268	28	7	0	3	4	edef	afad	edef
269	29	13	0	3	10	edef	afad	edef
270	30	4	0	0	4	edef	afad	edef

6.2.7 Analisi dei dati

Una volta raccolti, i dati sono stati analizzati ed i risultati dell'analisi sono stati inseriti nelle seguenti tabelle e riguardano le medie e le deviazioni standard dei valori scelti, dei punti ottenuti per turno o totali (punticum) e della somma

ottenuta dalle terne. Una delle tabelle è dedicata ai dati degli umani che hanno effettuato l'apprendimento con agenti altruisti (*afad*), l'altra con agenti egoisti (*edef*). Ogni tabella è composta da una parte riguardane la fase di apprendimento ed una di gioco tra umani.

***APPR=afad* PARTE=1**

Variable	*N*	*Mean*	*Std Dev*	*Minimum*	*Maximum*
SCELTA	180	2,933	1,646	0	10
PUNTI	180	30,122	10,978	-3	37
PUNTICUM	180	233,922	144,568	0	555
SOMMA	180	10,900	2,561	5	24

APPR=afad PARTE=2

Variable	*N*	*Mean*	*Std Dev*	*Minimum*	*Maximum*
SCELTA	180	2,611	1,962	0	10
PUNTI	180	16,944	18,108	-4	40
PUNTICUM	180	124,728	99,036	-9	387
SOMMA	180	8,806	3,719	0	24

APPR=edef PARTE=1

Variable	*N*	*Mean*	*Std Dev*	*Minimum*	*Maximum*
SCELTA	180	4,533	3,200	0	10
PUNTI	180	20,967	16,000	-4	37
PUNTICUM	180	154,894	137,819	-6	540
SOMMA	180	10,361	3,379	3	24

APPR=edef PARTE=2

Variable	*N*	*Mean*	*Std Dev*	*Minimum*	*Maximum*
SCELTA	180	3,400	2,826	0	10
PUNTI	180	15,711	17,532	-4	40
PUNTICUM	180	125,372	98,370	-9	393
SOMMA	180	9,228	4,189	0	24

Innanzitutto notiamo che i punti cumulati sono sempre superiori nella fase di apprendimento rispetto alla fase di gioco tra umani. Questo aspetto era stato previsto, infatti sembrava plausibile che il coordinamento fosse più semplice con gli agenti virtuali che con quelli umani. Tutti i partecipanti ai turni dall'1 al 4 non erano a conoscenza della fase di apprendimento con agenti virtuali, ed hanno trovato più semplice il gioco con la macchina rispetto a quello con gli umani. Infatti, al termine dell'esperimento molti dei giocatori ritenevano di aver giocato la prima fase con compagni di squadra più intelligenti e più disposti a cooperare; in realtà le strategie di gioco erano solo più semplici da intuire e da assecondare.

I dati delle tabelle indicano inoltre che ci sono differenze nel comportamento dei giocatori umani durante l'apprendimento con agenti virtuali altruisti o egoisti.

La fase di apprendimento dell'esperimento è stata introdotta per cercare di influenzare le strategie dei diversi giocatori; nelle nostre aspettative chi giocava

con agenti virtuali altruisti avrebbe dovuto adottare una strategia egoista e viceversa per chi giocava con egoisti. Dai dati infatti notiamo che la media delle scelte effettuate nella prima fase è decisamente più elevata (4,5) per chi ha giocato con gli egoisti rispetto a quella (2,9) di chi ha giocato con gli altruisti. I punti totali rispecchiano il comportamento tenuto, sono infatti più elevati quelli ottenuti con una strategia egoista.

Nella fase di gioco tra umani i valori si avvicinano molto riducendo la forbice che si era creata nella fase di apprendimento. Rimane invariata però la tendenza a giocare numeri più bassi se si è prima interagito con agenti virtuali altruisti, anche se non sembra una strategia decisamente vincente dai punti totali ottenuti (125 contro 124).

Le tabelle seguenti riportano la frequenza e la percentuale dei singoli valori delle somme, delle scelte e dei punti ottenuti. Anch'esse sono divise secondo il tipo di apprendimento e la parte (1ª o 2ª) dell'esperimento.

APPR=afad parte=1

SOMMA	*Frequency*	*Percent*	*Cumulative Frequency*	*Cumulative Percent*
5	1	0,6	1	0,6
7	11	6,1	12	6,7
8	4	2,2	16	8,9
10	103	57,2	119	66,1
11	30	16,7	149	82,8
14	12	6,7	161	89,4
15	5	2,8	166	92,2
17	13	7,2	179	99,4
24	1	0,6	180	100

SCELTA	*Frequency*	*Percent*	*Cumulative Frequency*	*Cumulative Percent*
0	19	10,6	19	10,6
1	10	5,6	29	16,1
3	116	64,4	145	80,6
4	30	16,7	175	97,2
10	5	2,8	180	100

PUNTI	*Frequency*	*Percent*	*Cumulative Frequency*	*Cumulative Percent*
-3	4	2,2	4	2,2
-1	5	2,8	9	5
0	7	3,9	16	8,9
20	5	2,8	21	11,7
26	30	16,7	51	28,3
27	9	5	60	33,3
29	5	2,8	65	36,1
30	12	6,7	77	42,8
37	103	57,2	180	100

APPR=afad parte=2

SOMMA	*Frequency*	*Percent*	*Cumulative Frequency*	*Cumulative Percent*
0	4	2,2	4	2,2
1	3	1,7	7	3,9
3	8	4,4	15	8,3
4	11	6,1	26	14,4
5	1	0,6	27	15
6	20	11,1	47	26,1
7	9	5	56	31,1
8	6	3,3	62	34,4
9	22	12,2	84	46,7
10	64	35,6	148	82,2
11	17	9,4	165	91,7
13	2	1,1	167	92,8
16	7	3,9	174	96,7
17	2	1,1	176	97,8
18	2	1,1	178	98,9
21	1	0,6	179	99,4
24	1	0,6	180	100

SCELTA	*Frequency*	*Percent*	*Cumulative Frequency*	*Cumulative Percent*
0	44	24,4	44	24,4
1	8	4,4	52	28,9
3	80	44,4	132	73,3
4	43	23,9	175	97,2
10	5	2,8	180	100

PUNTI	*Frequency*	*Percent*	*Cumulative Frequency*	*Cumulative Percent*
-3	40	22,2	46	25,6
-1	7	3,9	53	29,4
0	31	17,2	84	46,7
20	5	2,8	89	49,4
26	13	7,2	102	56,7
27	12	6,7	114	63,3
29	1	0,6	115	63,9
30	1	0,6	116	64,4
36	24	13,3	140	77,8
37	28	15,6	168	93,3
40	12	6,7	180	100

APPR=edef parte=1

SOMMA	*Frequency*	*Percent*	*Cumulative Frequency*	*Cumulative Percent*
3	7	3,9	7	3,9
4	1	0,6	8	4,4
5	1	0,6	9	5
6	11	6,1	20	11,1
7	21	11,7	41	22,8
8	1	0,6	42	23,3
9	9	5	51	28,3
10	72	40	123	68,3
11	4	2,2	127	70,6
13	20	11,1	147	81,7
14	17	9,4	164	91,1
15	1	0,6	165	91,7
16	6	3,3	171	95
17	8	4,4	179	99,4
24	1	0,6	180	100

SCELTA	*Frequency*	*Percent*	*Cumulative Frequency*	*Cumulative Percent*
0	24	13,3	24	13,3
1	2	1,1	26	14,4
3	42	23,3	68	37,8
4	72	40	140	77,8
10	40	22,2	180	100

PUNTI	*Frequency*	*Percent*	*Cumulative Frequency*	*Cumulative Percent*
-4	15	8,3	15	8,3
-3	20	11,1	35	19,4
-1	1	0,6	36	20
0	15	8,3	51	28,3
20	40	22,2	91	50,6
26	4	2,2	95	52,8
27	3	1,7	98	54,4
29	1	0,6	99	55
30	9	5	108	60
36	53	29,4	161	89,4
37	19	10,6	180	100

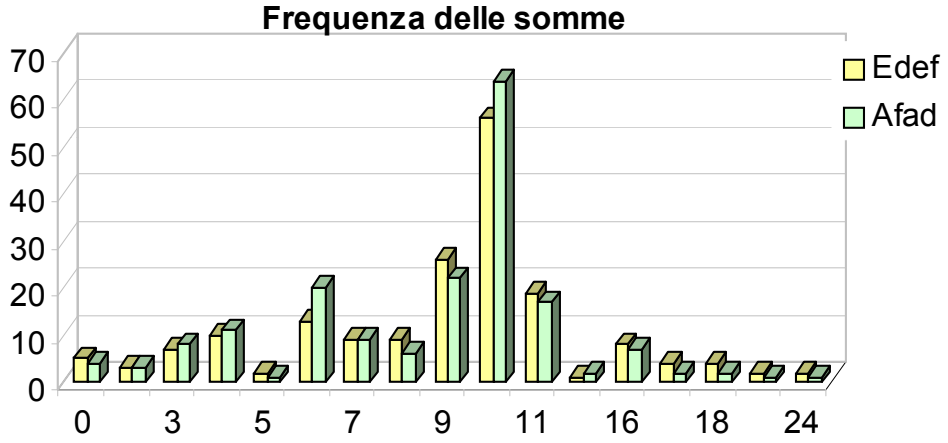
APPR=edef parte=2

SOMMA	Frequency	Percent	Cumulative Frequency	Cumulative Percent
0	5	2,8	5	2,8
1	3	1,7	8	4,4
3	7	3,9	15	8,3
4	10	5,6	25	13,9
5	2	1,1	27	15
6	13	7,2	40	22,2
7	9	5	49	27,2
8	9	5	58	32,2
9	26	14,4	84	46,7
10	56	31,1	140	77,8
11	19	10,6	159	88,3
13	1	0,6	160	88,9
16	8	4,4	168	93,3
17	4	2,2	172	95,6
18	4	2,2	176	97,8
21	2	1,1	178	98,9
24	2	1,1	180	100

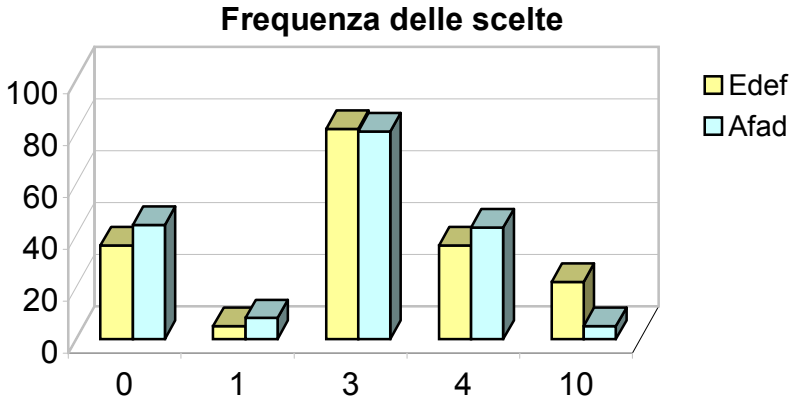
SCELTA	Frequency	Percent	Cumulative Frequency	Cumulative Percent
0	36	20	36	20
1	5	2,8	41	22,8
3	81	45	122	67,8
4	36	20	158	87,8
10	22	12,2	180	100

PUNTI	Frequency	Percent	Cumulative Frequency	Cumulative Percent
-4	16	8,9	16	8,9
-3	41	22,8	57	31,7
-1	4	2,2	61	33,9
0	23	12,8	84	46,7
20	10	5,6	94	52,2
26	16	8,9	110	61,1
27	12	6,7	122	67,8
29	1	0,6	123	68,3
30	13	7,2	136	75,6
36	4	2,2	140	77,8
37	28	15,6	168	93,3
40	12	6,7	180	100

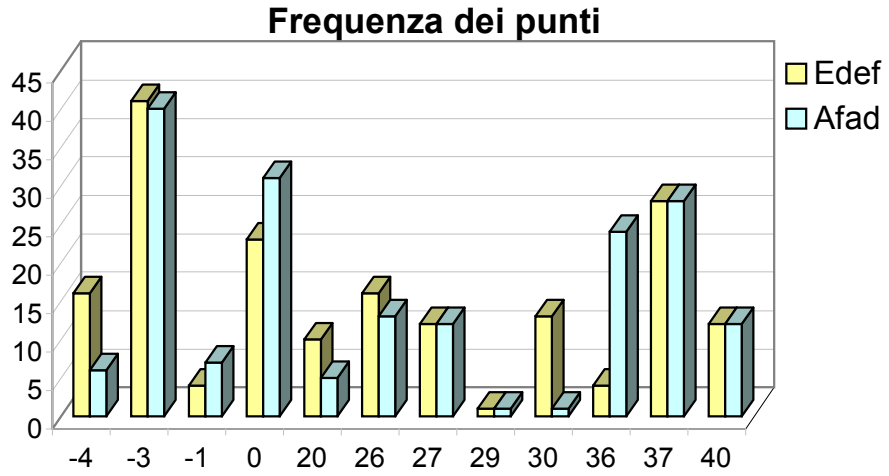
L'analisi delle frequenze delle somme appare smentire un'eventuale differenza tra i due gruppi di giocatori umani. Come risulta chiaro dal grafico seguente, la distribuzione delle somme è pressoché identica.



La frequenza delle scelte presenta la stessa situazione, si può notare però che il numero dei 10 è sensibilmente più elevato. Ciò dipende, senza dubbio, dall'apprendimento fatto (con agenti egoisti), ma è soprattutto espressione della strategia molto altruista di alcuni giocatori.



Infine dalla distribuzione dei punti per turno si può notare ancora una volta che le differenze sono veramente minime.



Riporto di seguito i grafici più significativi, riguardanti i singoli andamenti delle terne, per permettere una visione d'insieme delle strategie effettivamente riscontrate.

6.3 Il secondo esperimento di Torino

Poiché l'esperimento effettuato il 3 maggio del 2001 si era mostrato particolarmente interessante sotto molteplici punti di vista anche quest'anno abbiamo deciso di effettuarne uno, sempre con l'aiuto del dottor Novarese che come l'anno precedente si è occupato soprattutto dello sviluppo della parte logica del gioco. Nel 2001 erano comparsi alcuni problemi che desideravamo tentare di risolvere, in più i risultati ottenuti ci facevano ben sperare dandoci la convinzione che effettuando un nuovo gioco avremmo potuto raccogliere ulteriori dati per verificare le nostre ipotesi su altruismo, egoismo, cooperazione. Così si è deciso di apportare alcune modifiche al codice per variare parzialmente il gioco, si è stabilita una data e si è partiti con il reclutamento.

A differenza dell'anno precedente abbiamo notato una sempre maggior partecipazione e convinzione dei docenti nel pubblicizzare il nostro esperimento simulativo, ma forse un minor livello di adesione da parte degli studenti. Dopo aver conosciuto il numero approssimativo di coloro i quali intendevano aiutarci facendo da "cavie", abbiamo deciso quali modifiche apportare ai listati.

Innanzitutto si è optato per un maggiore numero di turni di apprendimento elevando tale fase a 26 giocate e per un minore numero di turni di gioco vero e proprio (interazione tra gli umani). Gli algoritmi degli agenti artificiali sono rimasti pressoché invariati, mentre abbiamo modificato il vettore di *payoff* disponibile: l'anno scorso ciascun individuo poteva scegliere tra 0, 1, 3, 4 e 10 quest'anno si è deciso di incrementare tali valori includendo tutti i numeri compresi tra lo 0 e il 10. Questa variazione è stata fatta per valutare come e se i giocatori, avendo un maggior ventaglio di scelte e quindi un maggior numero di combinazioni possibili per raggiungere il "famoso" 10, riuscivano a coordinarsi meglio e per valutare se emergevano comportamenti nettamente altruistici a vantaggio di altri puramente egoistici.

Anche in questa seconda esperienza abbiamo consegnato ai giocatori un foglio in cui inserire i propri dati ed il questionario, del tutto analogo a quello dell'anno precedente per valutare la comprensione delle regole, ma a differenza del 2001 tali regole non sono state spiegate verbalmente, ma trascritte su un foglio e consegnate agli studenti. Si è deciso di operare in questo modo per avere una maggiore omogeneità di informazione senza le possibili distorsioni che ne potrebbero derivare da una spiegazione orale, magari lievemente differente da un turno all'altro.

Al momento dell'esperimento abbiamo potuto contare sulla partecipazione di 36 persone, constatando un discretamente elevato "tasso di mortalità" dalla prima adesione sui fogli di reclutamento.

Prima di descrivere i fogli presentati agli studenti bisogna ancora segnalare l'ulteriore modifica al gioco: quest'anno non abbiamo utilizzato un metodo unico per il calcolo dei punteggi (40 per la terna che raggiunge il 10, 20 e non più 30 per il terzetto che supera con la somma delle proprie giocate il numero10, per far pagare un più caro prezzo al mancato 10, e 0 per chi sta sotto il 10). A questo metodo di calcolo, applicato a metà del numero dei partecipanti, ne è stato infatti affiancato uno nuovo. Ai valori 40, 20, 0 ora si è voluto sottrarre la media dei numeri dichiarati dalla terna.

Ecco il questionario compilato dai giocatori e i due differenti fogli in cui abbiamo spiegato le regole:

Torino, 19-4-2002

Istruzioni del gioco Somma 10

Ciascun giocatore viene abbinato dal computer ad altri due giocatori presenti in questa stanza.

Il gioco si svolge su più turni.

In ciascun turno, ogni componente del gruppo ha a disposizione il set di numeri compresi tra 0 e 10.

Il gioco consiste nel selezionare uno di questi numeri, sapendo che:

- i numeri dichiarati dai componenti della squadra saranno sommati;

- in base alla somma ottenuta, sarà attribuito un punteggio, secondo la seguente regola:

*se la somma dei tre numeri è esattamente pari a 10, ciascun giocatore riceve
40 punti - numero dichiarato

poniamo che un giocatore dichiari 10 e gli altri 3, la somma è 10; chi ha scelto 10, 40-10=30 punti, gli altri, avendo dichiarato 0, riceveranno 40-0=40 punti

* se la somma dei tre numeri è superiore a 10, ciascun giocatore riceve
20 punti - numero dichiarato

* se la somma dei tre numeri è inferiore a 10, ciascun giocatore riceve
0 punti - numero dichiarato (un punteggio negativo porta a sottrarre punti dal totale)

Al termine di ogni turno, il giocatore riceve l'informazione sui numeri giocati dagli altri due componenti della sua squadra, identificati stabilmente dai numeri 1 e 2.

I punti ottenuti in ogni turno sono sommati. Il proprio punteggio totale è mostrato sullo schermo.

L'abbinamento dei giocatori rimane stabile per un certo numero di turni; poi il computer procede ad un nuovo abbinamento casuale con altri due giocatori per un'altra sessione di turni.
Il nuovo abbinamento viene segnalato dal computer.

Entrambi gli abbinamenti sono e resteranno anonimi. Per garantire questa condizione, è possibile passare al turno successivo del gioco, solo quando tutti i giocatori hanno effettuato la propria scelta.

In base al punteggio totale sarà stilata una classifica di tutti i 36 giocatori impegnati oggi e saranno assegnati i seguenti premi:
- dal primo al quarto:buono libro del valore di 20€
- dal quinto all'ottavo: buono libro del valore di 15€
- dal nono al dodicesimo: buono libro del valore di 10€
- dal tredicesimo al sedicesimo: buono libro del valore di 5€

- dal diciassettesimo al ventottesimo: buono libro del valore di 3€

Al termine del gioco compilare il questionario allegato
Non è consentito prendere appunti su fogli.

Torino, 19-4-2002

Istruzioni del gioco Somma 10

Ciascun giocatore viene abbinato dal computer ad altri due giocatori presenti in questa stanza.

Il gioco si svolge su più turni.

In ciascun turno, ogni componente del gruppo ha a disposizione il set di numeri compresi tra 0 e 10.

Il gioco consiste nel selezionare uno di questi numeri, sapendo che:

- i numeri dichiarati dai componenti della squadra saranno sommati,

- in base alla somma ottenuta, sarà attribuito un punteggio, secondo la seguente regola:

*se la somma dei tre numeri è esattamente pari a 10, ciascun giocatore riceve
40 punti - media dei numeri dichiarati dai tre componenti della squadra

poniamo che 2 giocatori dichiarino 3 e uno 4, la somma è 10, la media dei numeri dichiarati è (3+3+4)/3=3,3

* se la somma dei tre numeri è superiore a 10, ciascun giocatore riceve
20 punti - media dei numeri dichiarati dai tre componenti della squadra

* se la somma dei tre numeri è inferiore a 10, ciascun giocatore riceve
0 punti - media dei numeri dichiarati dai tre componenti della squadra (un punteggio negativo porta a sottrarre punti dal totale)

Al termine di ogni turno, il giocatore riceve l'informazione sui numeri giocati dagli altri due componenti della sua squadra, identificati stabilmente dai numeri 1 e 2.

I punti ottenuti in ogni turno sono sommati. Il proprio punteggio totale è mostrato sullo schermo.

L'abbinamento dei giocatori rimane stabile per un certo numero di turni; poi il computer procede ad un nuovo abbinamento casuale con altri due giocatori per un'altra sessione di turni.
Il nuovo abbinamento viene segnalato dal computer.

Entrambi gli abbinamenti sono e resteranno anonimi. Per garantire questa condizione, è possibile passare al turno successivo del gioco, solo quando tutti i giocatori hanno effettuato la propria scelta.

In base al punteggio totale sarà stilata una classifica di tutti i 36 giocatori impegnati oggi e saranno assegnati i seguenti premi:
- dal primo al quarto:buono libro del valore di 20€
- dal quinto all'ottavo: buono libro del valore di 15€
- dal nono al dodicesimo: buono libro del valore di 10€
- dal tredicesimo al sedicesimo: buono libro del valore di 5€
- dal diciassettesimo al ventottesimo: buono libro del valore di 3€

Al termine del gioco compilare il questionario allegato
Non è consentito prendere appunti su fogli.

Computer numero: …..
turno: …. aula: ….

Nome e cognome:
……

data di nascita:
……

Facoltà:
……

matricola:
……

e-mail: ……

recapito telefonico: ……

****
Stai giocando a somma 10,
- sai con certezza che la somma dei numeri giocati dalle altre persone nel tuo gruppo è pari a 10, che numero giochi?

- sai con certezza che la somma dei numeri giocati dalle altre persone nel tuo gruppo è pari a 0, che numero giochi?

- sai con certezza che la somma dei numeri giocati dalle altre persone nel tuo gruppo è pari a 6, che numero giochi?

- sai con certezza che la somma dei numeri giocati dalle altre persone nel tuo gruppo è pari a 7, che numero giochi?

I dati ottenuti saranno trattati unicamente in forma anonima (dall'Università di Torino e dall'Università del Piemonte Orientale) ed utilizzati a fini scientifici e per la distribuzione dei premi.

Autorizzo il trattamento dei dati in conformità alla legge 675/'96

Riassumendo l'architettura del gioco è la seguente:

STRUTTURA DIVISA IN DUE PARTI

Parte prima: 26 turni: ogni umano gioca con due agenti artificiali

(aumenta la lunghezza della fase di apprendimento)

Parte seconda (9 turni): gli agenti umani giocano tra loro

GLI AGENTI ARTIFICIALI

- gruppo A (18 o 21 persone): gioca con gli agenti egoisti (debole e forte, caratterizzati come nell'esperimento dell'anno scorso),
- gruppo B (18 o 21): gioca con gli agenti altruisti (di nuovo come quelli dell'anno scorso)

(per il futuro: si potrebbe pensare a due agenti "equi" che giocano

rispettivamente sempre in sequenza: 433 e 343)

I SISTEMI DI REMUNERAZIONE

- gruppo A1 (metà del gruppo A) e B1 (metà del gruppo B): vincite definite individualmente e pari a:

- se somma =10; 40- numero dichiarato
- se somma>10; 20-numero di dichiarato (l'anno scorso era 30-numero dichiarato, ma ora penalizziamo maggiormente questo risultato)
- se somma<10; 0- numero dichiarato

- gruppo A2 e B2: vincite uguali per tutto il gruppo e pari a:

come per A1 e B1, ma invece del numero dichiarato da ciascuno si sottrae la media dei numeri dichiarati dai tre approssimata all'intero più vicino)

6.3.1 Modifiche al codice

1) Variando il vettore dei *payoff* è stato indispensabile effettuare alcuni cambiamenti relativi all'impostazione grafica dell'*applet*.
   - Modifica a *Sum10Client.java*

Abbiamo eliminato le *textfield*, le *label* e le *checkbox* (elementi grafici in cui comparivano i 5 possibili valori tra cui scegliere) e abbiamo inserito il menù a tendina contenente tutti i numeri compresi tra 0 e 10. Sempre relativamente ai differenti *payoff* abbiamo cancellato l'*array* (vettore) che veniva caricato dal *server* e che conteneva i valori 0, 1, 3, 4, 10

```
if (valueS == "A")
value = payoffArray[0];
     if (valueS == "B")
value = payoffArray[1];
     if (valueS == "C")
     value = payoffArray[2];
     if (valueS == "D")
     value = payoffArray[3];
     if (valueS == "E")
     value = payoffArray[4];
```

e l'abbiamo sostituito con

```
value=CBScelta.getSelectedIndex()
```

stringa che permette di prendere automaticamente il valore scelto dall'utente.

2) Per suddividere l'esperimento nelle 4 parti, A1, B1, A2, B2 è stato indispensabile operare sul file *RuleMaster.java*

   - Con la seguente stringa è stato possibile operare tale suddivisione

```
if((a.id%3==1) && (a.id<54))
```

che significa: se il resto della divisione per 3 è uguale a 1 (4,7,10,…) e se sono nei primi 54 giocatori, allora interagisco con l'egoista debole

```
if((a.id%3==2) && (a.id<54))
```

se il resto e' uguale a 2 (5,8,11) e sono nei primi 54 con l'egoista forte

```
if((a.id%3==1) && (a.id>54))
```

che significa: se il resto della divisione per 3 è uguale a 1 (4,7,10,…) e se sono nei secondi 54 giocatori, allora gioco con l'altruista debole

```
if((a.id%3==2) && (a.id>54))
```

se il resto e' uguale a 2 (5,8,11) e sono nei secondi 54 con l'altruista forte.

3) Per cambiare il calcolo dei punteggi abbiamo modificato il file *Player.java*

- Modifica del calcolo dei punteggi:

Il gruppo A1 è definito così:

```
(this.id<27)
```

il gruppo A2:

```
(this.id>=27 && this.id<54)
```

il B1:

```
(this.id>=54 && this.id<81)
```

il B2:

```
(this.id>=81)
```

```
if((this.id<27) || (this.id>=54 && this.id<81))
   {
        sum3 = getSum3();
        if (sum3 == 10)
```

```
            points = 40 - choice;
        if (sum3 > 10)
            points = 20 - choice;
        if (sum3 < 10)
        points = - choice;
    }
    if((this.id>=27 && this.id<54) || (this.id>=81))
    {
        sum3 = getSum3();
        if (sum3 == 10)
            points = 40 - Math.round(sum3/3);
        if (sum3 > 10)
            points = 20 - Math.round(sum3/3);
        if (sum3 < 10)
            points = - Math.round(sum3/3);
```

4) Abbiamo adattato il *Model Swarm* al nuovo numero di cicli e di giocatori agendo sul *Sum10Model Swarm.java*

   - Ora il numero di giocatori è 108 e il numero di cicli 35 (26 di apprendimento e 9 di gioco vero e proprio)

```
numPlayers = 108;
numCycles = 35;
```

5) E in ultimo abbiamo cambiato il numero di turni di apprendimento variando il file Timer.java

```
if ((cycles + 1) == 26)
```

(si impone che dopo 26 cicli su 35 si incominci a giocare tra gli umani).

6.3.2 Altri problemi incontrati

Purtroppo l'idea che ci eravamo fatti l'anno scorso, imputando l'impossibilità di effettuare le simulazioni con un elevato numero di partecipanti alla lentezza dei *client*, computer su cui operavano i giocatori, si è dimostrata errata. Quest'anno, infatti, i pc delle aule del Liases erano appena stati integralmente cambiati e si disponeva quindi di

macchine molto più potenti. Durante la fase di test, svolta da me con l'aiuto di Alessandro Cappellini, studente tesista, abbiamo provato a connettere i 36 pc necessari per l'esperimento con il *server* su cui avrebbe dovuto operare la simulazione, ma contro ogni aspettativa, abbiamo riscontrato il medesimo problema dell'anno precedente: al massimo siamo riusciti a far interagire 9 macchine con una buona stabilità, ma non le 36 desiderate. Si sono allora rese indispensabili altre considerazioni e modifiche. Innanzitutto, date le due esperienze negative con più di 9 pc collegati in rete, il problema sarà da ricercarsi nell'architettura del software Swiee, forse l'utilizzo della classe Rmi (chiamate ai metodi remoti) non è la migliore soluzione, anche perché la stessa Sun, casa produttrice di Java ha poi ammesso l'eccessiva lentezza di tali classi, oppure, tali difficoltà di comunicazione tra le macchine potrebbe derivare da una non lineare scrittura degli algoritmi generati per introdurre gli agenti artificiali.

Dato però lo scarso tempo a disposizione, abbiamo deciso, in accordo con il professor Terna, col dottor Novarese e con dottor Boero (ideatore di Swiee) di intervenire sul codice della simulazione "spezzandola" in quattro diverse, al fine di mantenere inalterata la struttura proposta dal dottor Novarese e la suddivisione nei quattro gruppi (A1, B1, A2, B2). Praticamente si è trattato di scindere il codice precedentemente programmato come vedremo di seguito:

- Prendiamo in considerazione solo il file relativo alla generazione degli agenti artificiali[24], *RuleMaster.java*. Prima un solo file conteneva i 4 diversi tipi di agenti artificiali (altruista forte e debole, egoista forte e debole), ora abbiamo dovuto creare 2 diversi file, uno solo relativo agli agenti altruisti e l'altro per quelli egoisti.

[24] Teniamo comunque presente che è stato necessario modificare i file: *Player.java* (per avere 2 differenti simulazioni con i due differenti metodi per il calcolo dei punteggi), *Timer. java* (per ridurre il numero di giocatori per ciascuna simulazione) e *RemoteServer.java* (per variare l'indirizzo IP, avendo più server le simulazioni devono contenere l'indirizzo specifico della macchina su cui girano.

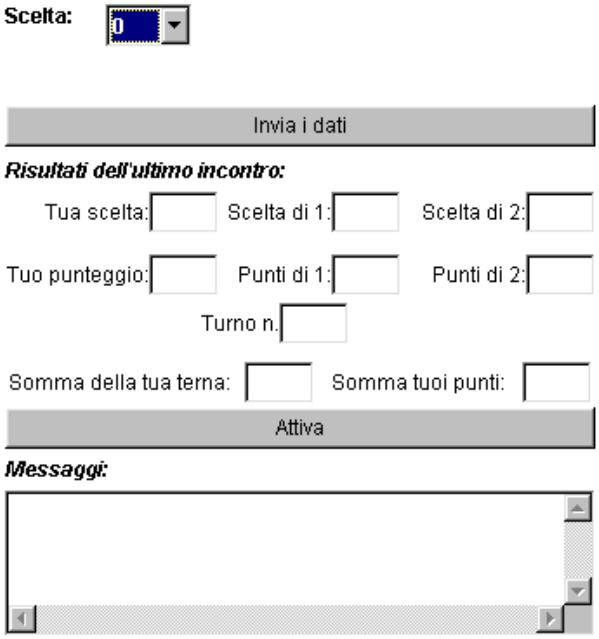
```
if(a.id%3==1)
{
	//egoista debole

	if (timer.cycles == 0)
	{
		b = 3;
	}
	else
	{
		if(((a.playerArray[a.id1].choice2+a.playerArray[
		a.id2].choice2)<7||(a.playerArray[a.id1].choice2
		+ a.playerArray[a.id2].choice2) > 10))
		{
			b = 0;
		}
		else
			b = 3;
	}
}
if(a.id%3==2)
{
	//egoista forte
	b = 3;
}
return b;
```

Una volta suddivisa la simulazione in quattro abbiamo deciso di utilizzare due *server* e di fare due turni: su un *server* ha operato la simulazione con agenti altruisti e con il nuovo metodo per il calcolo dei punteggi (la media dei valori giocati dalla terna) e sull'altro l'esperimento con agenti egoisti e sempre col nuovo metodo di calcolo della media. A ciascun *server* abbiamo collegato i 9 pc, quindi in ogni turno avevamo 18 giocatori umani contemporaneamente. Nel secondo turno abbiamo caricato i due codici

in cui il punteggio veniva calcolato con il "vecchio metodo" (40 meno il numero dichiarato da ciascuno oppure 20 o 0). Così facendo non abbiamo modificato alcuna delle variabili ambientali che avrebbero potuto influenzare l'analisi dei risultati. In tutto hanno giocato infatti i 36 giocatori previsti e la struttura a 4 gruppi ideata dal dottor Novarese è rimasta inalterata.

La finestra contenente l'*applet* che ciascun giocatore trovava visualizzata sul proprio pc è la seguente:



Come l'anno scorso, ciascun giocatore all'inizio dell'esperimento doveva premere il tasto "Attiva" in modo da connettere il *client* con il *server*, poi effettuare la scelta aprendo il menù a tendina e inviare i dati col tasto "Invia i dati". Dopo aver inviato all'elaboratore la propria giocata, il tasto "Attiva" cambiava denominazione e compariva la scritta "Aggiorna i risultati". Nelle relative caselle apparivano poi le scelte effettuate dai tre membri della terna e i rispettivi punteggi, oltre che il numero del turno di gioco.

Quest'anno si è inoltre deciso di non comunicare a priori il numero di turni della simulazione per valutare se si creava una maggior competizione e per indagare sull'effetto sorpresa. Nessun partecipante al gioco era a conoscenza di interagire con agenti artificiali e come l'anno passato molti, al termine dei 35 turni, hanno osservato come nei primi 26 fosse stato più semplice trovare un equilibrio.

Nella tabella che segue riporto il modo in cui abbiamo deciso di destinare le vincite (come l'anno passato buoni libro da ritirarsi entro un mese presso la libreria all'interno della Facoltà):

Conti:

Valore del buono libro (in euro)	Numero buoni	Spesa totale
20	4	80
15	4	60
10	4	40
5	4	20
3	16	48
Totale		248

6.4 Analisi dei risultati

Anche per il secondo esperimento, dopo aver raccolto i dati li abbiamo elaborati ed interpretati, tentando di estrapolare il maggior numero di informazioni possibili. L'esame di tali dati è stato effettuato rispecchiando la suddivisione in due fasi del gioco per valutare quanto e se il *training* abbia effettivamente inciso sulla parte di gioco tra i soli umani.

Tabella 1

Punteggio	Dati	Altruisti	Egoisti	Totale
Punteggio nuovo	Media	**791.1**	**874.6**	832.8
	Deviazione standard	168.8	89.0	141.3
Punteggio vecchio	Media	**705.9**	**397.4**	551.7
	Deviazione standard	241.5	301.5	313.7
Media totale		748.5	636.0	692.3
Deviazione standard totale		212.7	326.1	281.0

Per quanto riguarda la prima parte dell'esperimento si possono notare innanzitutto tre elementi chiave, che appaiono evidenti dalla Tabella 1:

- Se il punteggio viene calcolato nel modo "vecchio" (se la somma dei tre numeri è esattamente pari a 10, ciascun giocatore riceve 40

punti - numero dichiarato) e se il giocatore interagisce con agenti egoisti ottiene in media le performance peggiore

- Il nuovo metodo per il calcolo del punteggio (se la somma dei tre numeri è esattamente pari a 10, ciascun giocatore riceve 40 punti - media dei numeri dichiarati dai tre componenti della squadra) garantisce un risultato medio più elevato
- Se si adotta il vecchio metodo per il calcolo dei punteggi e si considera chi ha appreso con agenti egoisti emerge che i punteggi medi ottenuti sono nettamente inferiori.

Inoltre si possono notare altre due caratteristiche: considerando il nuovo metodo di calcolo del punteggio, chi gioca con agenti egoisti ottiene un punteggio medio più alto con una deviazione standard estremamente bassa; in teoria, chi gioca con gli agenti altruisti, dovrebbe essere favorito dal punteggio calcolato nel modo "vecchio" e invece il punteggio medio è maggiore per il nuovo metodo di calcolo.

Il punteggio ottenuto dai giocatori dipende fondamentalmente da due fattori: dalla generosità e/o l'altruismo proprio e degli altri componenti del team (con il punteggio nuovo questo aspetto non dovrebbe contare) e dalla capacità di coordinarsi propria e degli altri componenti del team (e quindi alla capacità di arrivare a 10 o almeno ad un numero superiore). Ovviamente entrambi gli aspetti sono legati alla comprensione del gioco che anche quest'anno si è rivelata molto alta e forse quasi maggiore rispetto a quella rilevata lo scorso anno in base alle risposte alle domande del questionario; pochissimi giocatori sembrano infatti non averne capito la logica.

Distribuzione dei numeri scelti relativa alla prima fase di gioco

Punteggio calcolato nel modo"vecchio"

Tabella 2

	Training con altruisti									**Totale**
Scelta	Giocatore 2	7	14	15	21	25	29	33	36	
0	53.8%	0.0%	38.5%	0.0%	0.0%	3.8%	0.0%	7.7%	0.0%	11.5%
1	15.4%	0.0%	7.7%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	2.6%
2	0.0%	0.0%	3.8%	0.0%	0.0%	3.8%	0.0%	0.0%	0.0%	0.9%
3	0.0%	96.2%	0.0%	92.3%	92.3%	11.5%	65.4%	73.1%	92.3%	58.1%
4	0.0%	3.8%	0.0%	7.7%	7.7%	3.8%	30.8%	19.2%	7.7%	9.0%
5	7.7%	0.0%	7.7%	0.0%	0.0%	15.4%	3.8%	0.0%	0.0%	3.8%
6	19.2%	0.0%	19.2%	0.0%	0.0%	23.1%	0.0%	0.0%	0.0%	6.8%
7	3.8%	0.0%	0.0%	0.0%	0.0%	19.2%	0.0%	0.0%	0.0%	2.6%
8	0.0%	0.0%	0.0%	0.0%	0.0%	11.5%	0.0%	0.0%	0.0%	1.3%
9	0.0%	0.0%	3.8%	0.0%	0.0%	3.8%	0.0%	0.0%	0.0%	0.9%
10	0.0%	0.0%	19.2%	0.0%	0.0%	3.8%	0.0%	0.0%	0.0%	2.6%
Totale	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%

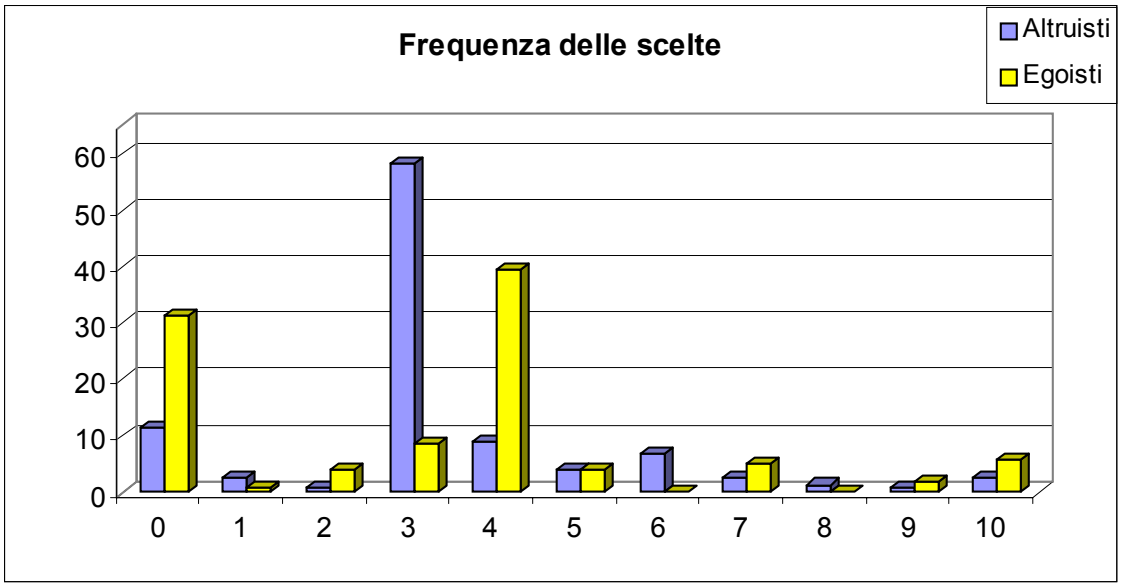
La Tabella 2 riporta le scelte effettuate dai singoli giocatori e il totale. Come si può notare il numero più scelto è 3 (quello che il sistema cerca di "imporre"). Ci sono però giocatori che offrono in una percentuale rilevante di casi numeri superiori a 3 (potrebbero essere scelte legate, oltre che alla mancata comprensione del gioco, alla volontà di non approfittare degli altri e di reciprocare il loro comportamento) ma anche giocatori che non si accontentano e offrono 0 (anche per molto più che la metà dei turni).

Tabella 3

	Training con egoisti									Totale
Scelta	1	6	10	18	19	20	24	26	35	
0	76.9%	0.0%	0.0%	46.2%	0.0%	76.9%	53.8%	19.2%	7.7%	31.2%
1	0.0%	0.0%	0.0%	3.8%	0.0%	0.0%	0.0%	0.0%	3.8%	0.9%
2	0.0%	0.0%	0.0%	0.0%	34.6%	0.0%	0.0%	0.0%	0.0%	3.8%
3	7.7%	0.0%	3.8%	15.4%	3.8%	7.7%	0.0%	30.8%	7.7%	8.5%
4	3.8%	100.0%	92.3%	11.5%	42.3%	0.0%	26.9%	50.0%	26.9%	39.3%
5	0.0%	0.0%	3.8%	3.8%	0.0%	0.0%	3.8%	0.0%	23.1%	3.8%
6	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
7	7.7%	0.0%	0.0%	11.5%	11.5%	0.0%	7.7%	0.0%	7.7%	5.1%
8	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
9	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	15.4%	1.7%
10	3.8%	0.0%	0.0%	7.7%	7.7%	15.4%	7.7%	0.0%	7.7%	5.6%
Totale	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%

Quasi tutti i giocatori si adattano a giocare 4, ma c'è una percentuale più elevata di 0, probabilmente per punire i partner troppo egoisti.

I dati raccolti nelle tabelle 1 e 2 sono ben visibili e riassunti dal grafico seguente.

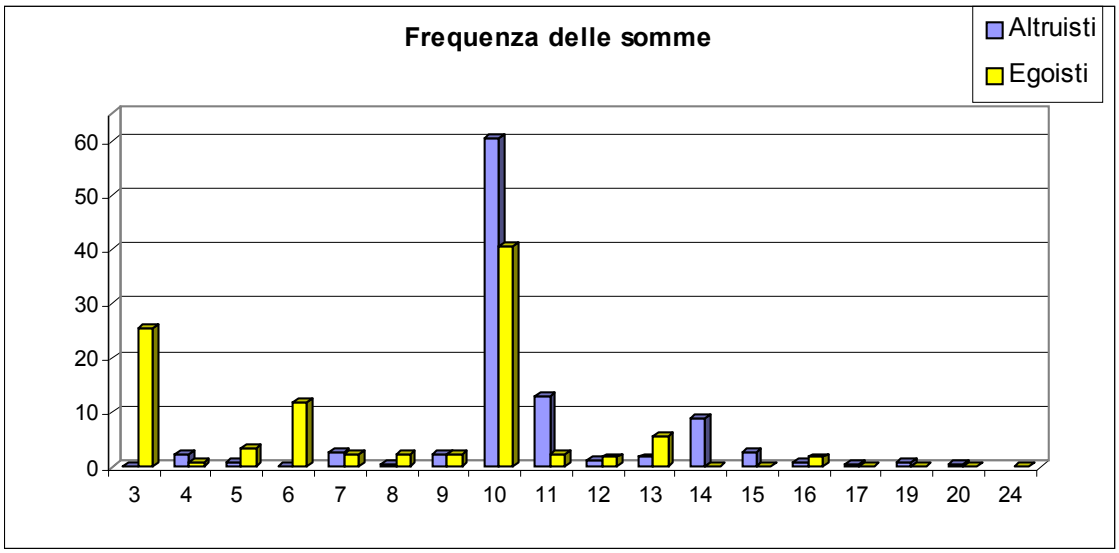


Chi gioca con agenti altruisti ottiene 10 in una percentuale sensibilmente più elevata di casi (come prevedibile).

Tabella 4

Somma terna	Altruisti	Egoisti	Totale complessivo
3	0.00%	25.64%	12.82%
4	2.14%	0.85%	1.50%
5	0.85%	3.42%	2.14%
6	0.00%	11.97%	5.98%
7	2.56%	2.14%	2.35%
8	0.43%	2.14%	1.28%
9	2.14%	2.14%	2.14%
10	60.68%	40.60%	50.64%
11	12.82%	2.14%	7.48%
12	1.28%	1.71%	1.50%
13	1.71%	5.56%	3.63%
14	8.97%	0.00%	4.49%
15	2.56%	0.00%	1.28%
16	0.85%	1.71%	1.28%
17	0.43%	0.00%	0.21%
19	0.85%	0.00%	0.43%
20	0.43%	0.00%	0.21%
24	1.28%	0.00%	0.64%
Totale complessivo	100.00%	100.00%	100.00%

La Tabella 4 è illustrata graficamente nell'istogramma seguente:



Punteggio calcolato utilizzando la media (metodo "nuovo")

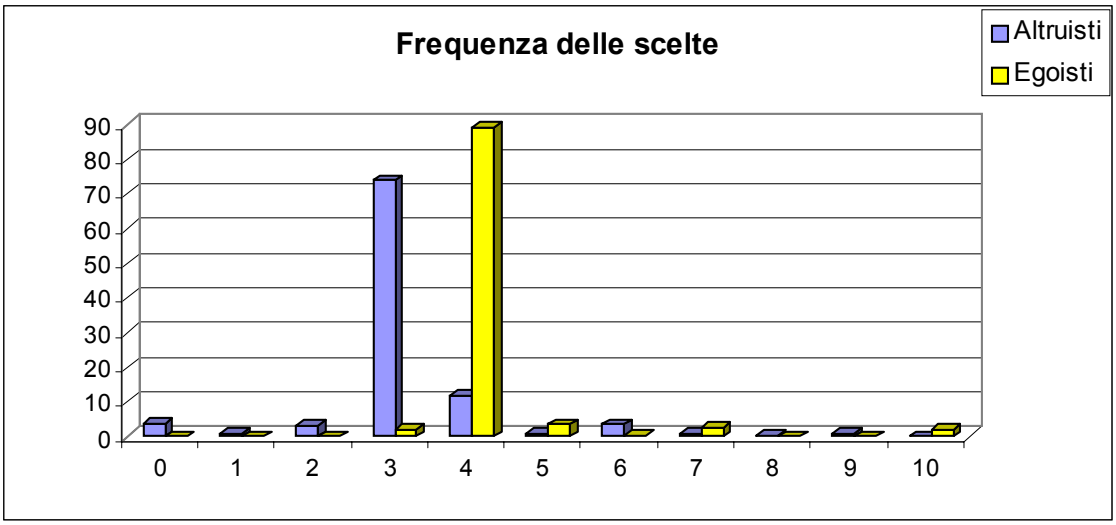
Tabella 5

	Training con altruisti									**Totale**
Scelta	3	4	11	16	23	27	28	31	32	
0	19.2%	15.4%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	3.8%
1	7.7%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.9%
2	15.4%	0.0%	0.0%	0.0%	3.8%	0.0%	7.7%	0.0%	0.0%	3.0%
3	34.6%	23.1%	73.1%	96.2%	76.9%	96.2%	80.8%	100.0%	84.6%	73.9%
4	23.1%	19.2%	26.9%	3.8%	11.5%	3.8%	7.7%	0.0%	11.5%	12.0%
5	0.0%	0.0%	0.0%	0.0%	3.8%	0.0%	3.8%	0.0%	0.0%	0.9%
6	0.0%	26.9%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	3.8%	3.4%
7	0.0%	3.8%	0.0%	0.0%	3.8%	0.0%	0.0%	0.0%	0.0%	0.9%
8	0.0%	3.8%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.4%
9	0.0%	7.7%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.9%
10	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
Totale	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%

Tabella 6

	Training con egoisti									**Totale**
Scelta	5	8	9	12	13	17	22	30	34	
0	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
1	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
2	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
3	0.0%	11.5%	0.0%	0.0%	3.8%	0.0%	0.0%	3.8%	0.0%	2.1%
4	96.2%	76.9%	92.3%	96.2%	88.5%	100.0%	92.3%	65.4%	96.2%	89.3%
5	3.8%	7.7%	3.8%	3.8%	7.7%	0.0%	3.8%	0.0%	0.0%	3.4%
6	0.0%	0.0%	3.8%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.4%
7	0.0%	3.8%	0.0%	0.0%	0.0%	0.0%	3.8%	15.4%	0.0%	2.6%
8	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
9	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
10	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	15.4%	3.8%	2.1%
Totale	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%

In questo caso le scelte sono molto più concentrare e raramente si opta per lo 0. La concentrazione è leggermente maggiore nel caso del training con agenti altruisti, ma dipende quasi esclusivamente dal comportamento di due giocatori (3 e 4 che durante la fase di training scelgono rispettivamente nel 19.2 e nel 15.4% dei casi il valore 0). Il grafico relativo alla frequenza delle scelte mostra quanto affermato.

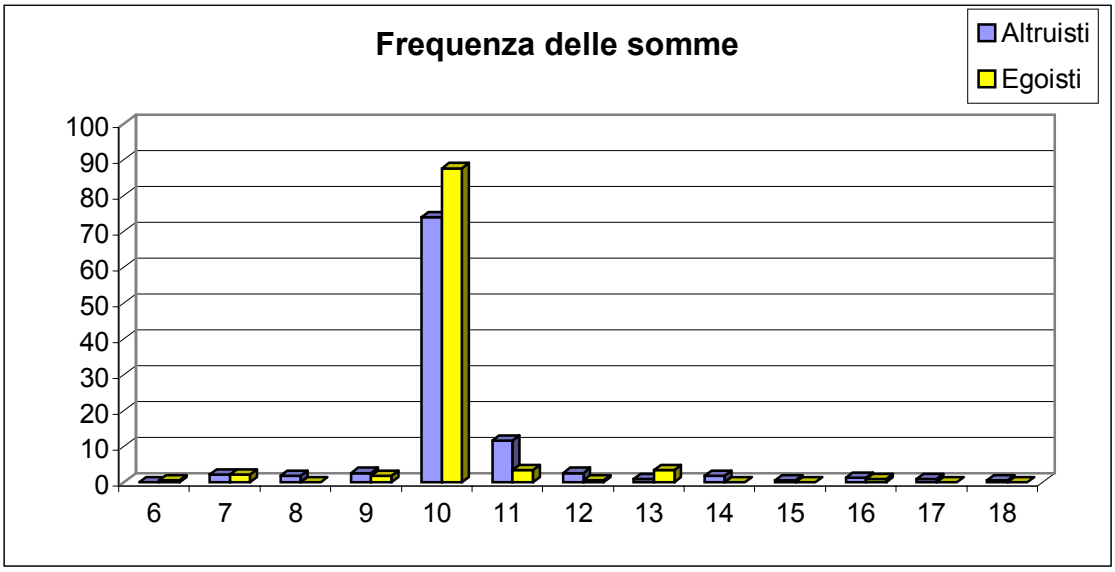


Ne deriva una maggiore percentuale di 10 ottenuti dal gruppo che gioca con egoisti.

Tabella 7

Somma terna	Altruisti	Egoisti	Totale complessivo
6	0.00%	0.43%	0.21%
7	2.14%	2.14%	2.14%
8	1.71%	0.00%	0.85%
9	2.56%	1.71%	2.14%
10	73.93%	87.61%	80.77%
11	11.54%	3.42%	7.48%
12	2.56%	0.43%	1.50%
13	0.85%	3.42%	2.14%
14	1.71%	0.00%	0.85%
15	0.43%	0.00%	0.21%
16	1.28%	0.85%	1.07%
17	0.85%	0.00%	0.43%
18	0.43%	0.00%	0.21%
Totale complessivo	100.00%	100.00%	100.00%

La Tabella 7 è illustrata nel grafico relativo alla frequenza delle somme seguente:



E' importante confrontare chi ha giocato con agenti altruisti valutando il peso dei due diversi sistemi per il calcolo del punteggio. La differente performance evidenziata nella Tabella 1 dipende da due fattori:

- dal tentativo di alcuni giocatori di offrire 0 (*free riding* massimo, forse indotto dal fatto di giocatore con agenti altruisti)
- ma anche dallo sforzo di alcuni giocatori di replicare in maniera coerente alle scelte degli altri (e di non essere quindi *free riders*).

Con il metodo di calcolo del punteggio "vecchio" emerge infatti:

- una percentuale molto più alta di giocatori che scelgono 0, ma anche
- una percentuale molto più alta di giocatori che scelgono numeri superiori a 3 e a 4
- e complessivamente, con i due sistemi di calcolo del punteggio, risulta una percentuale quasi analoga di turni in cui la somma ottenuta è maggiore o uguale a 10.

Tabella 8

Scelta	Punteggio "nuovo"	Punteggio "vecchio"
0	3.8%	11.5%
1	0.9%	2.6%
2	3.0%	0.9%
3	73.9%	58.1%
4	12.0%	9.0%
5	0.9%	3.8%
6	3.4%	6.8%
7	0.9%	2.6%
8	0.4%	1.3%
9	0.9%	0.9%
10	0.0%	2.6%
Totale	100.0%	100.0%

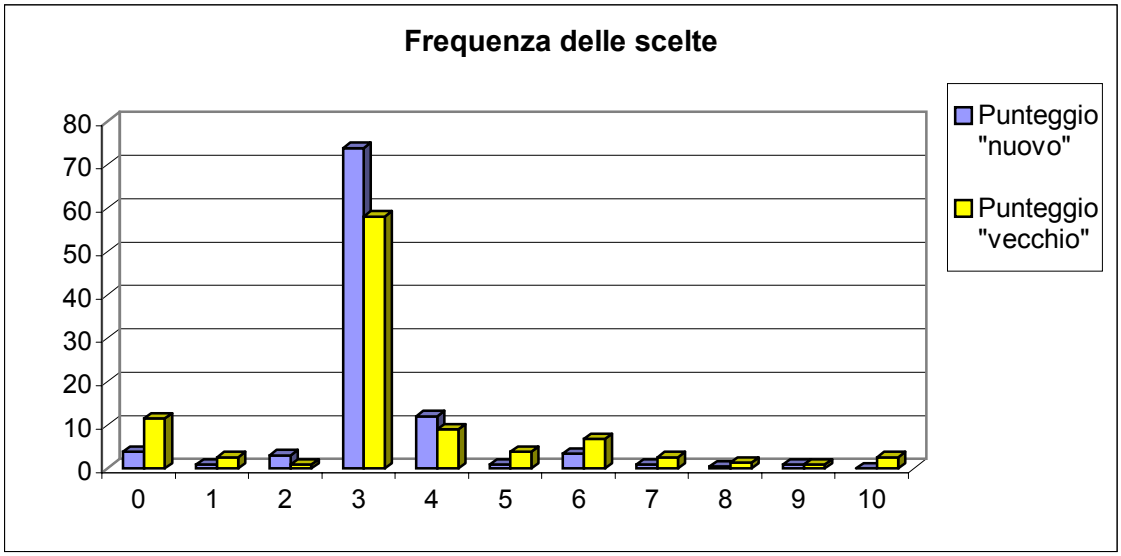
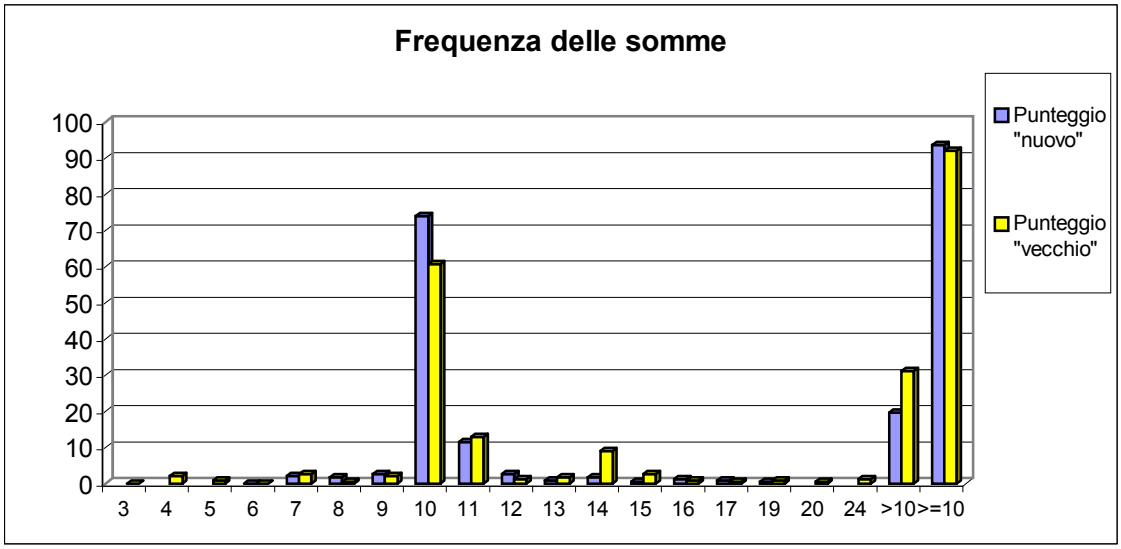


Tabella 9

Training con altruisti

Somma	Punteggio "nuovo"	Punteggio "vecchio"
3		0.00%
4		2.14%
5		0.85%
6	0.00%	0.00%
7	2.14%	2.56%
8	1.71%	0.43%
9	2.56%	2.14%
10	73.93%	60.68%
11	11.54%	12.82%
12	2.56%	1.28%
13	0.85%	1.71%
14	1.71%	8.97%
15	0.43%	2.56%
16	1.28%	0.85%
17	0.85%	0.43%
19	0.43%	0.85%
20		0.43%
24		1.28%
>10	**19.6**	**31.2**
≥10	**93.6**	**92.0**



Passiamo ora all'analisi della seconda fase del gioco, ovvero quella relativa all'interazione tra i soli umani.

Tabella 10

Punteggio		Altruisti	Egoisti	Totale
Punteggio "nuovo"	Media	**299.2**	**285.4**	292.3
	Deviazione standard	38.0	39.6	38.3
Punteggio "vecchio"	Media	**125.4**	**146.2**	135.8
	Deviazione standard	63.0	73.2	67.1
Media totale		**212.3**	**215.8**	214.1
Deviazione standard totale		102.7	91.6	95.9

Dato il calcolo del punteggio con il metodo della differenza tra il risultato ottenuto e il numero giocato, chi ha svolto il training con gli agenti egoisti ha ottenuto un punteggio medio più elevato. Con il nuovo calcolo del punteggio basato sulla media, la situazione si ribalta (e si ribalta anche rispetto alla prima parte analizzata).

Distribuzione dei numeri scelti relativi alla seconda fase di gioco

Punteggio calcolato utilizzando la media (metodo "nuovo")

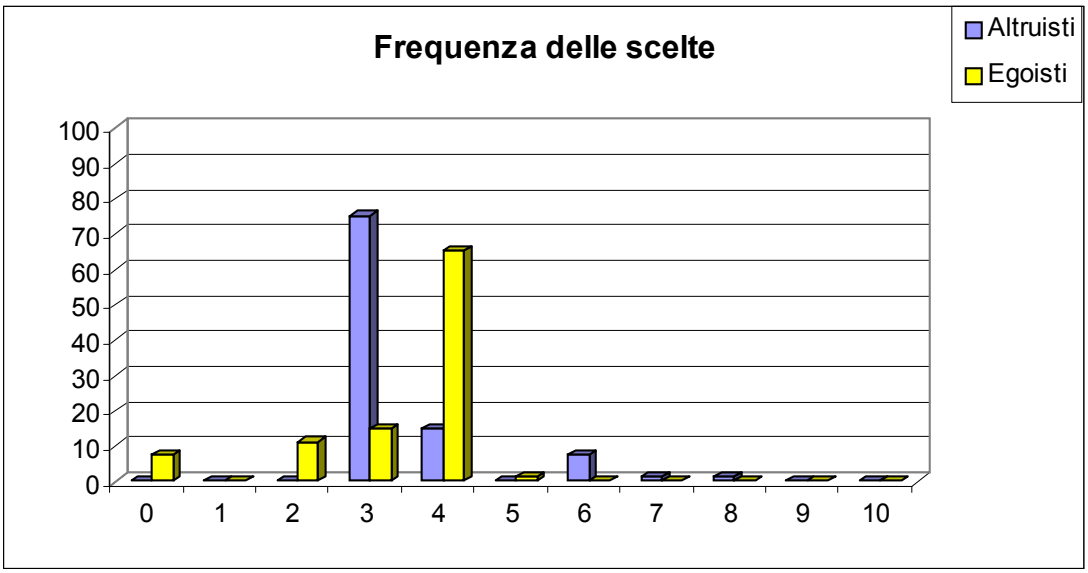
Tabella 11

	Training con altruisti									**Totale**
Scelta	3	4	11	16	23	27	28	31	32	
0	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
1	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
2	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
3	77.8%	11.1%	100.0%	100.0%	100.0%	100.0%	100.0%	88.9%	0.0%	75.3%
4	22.2%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	11.1%	100.0%	14.8%
5	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
6	0.0%	66.7%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	7.4%
7	0.0%	11.1%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	1.2%
8	0.0%	11.1%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	1.2%
9	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
10	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
Totale	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%

Tabella 12

	Training con egoisti									**Totale**
Scelta	5	8	9	12	13	17	22	30	34	
0	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	66.7%	0.0%	0.0%	7.4%
1	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
2	88.9%	0.0%	0.0%	11.1%	0.0%	0.0%	0.0%	0.0%	0.0%	11.1%
3	0.0%	88.9%	0.0%	0.0%	0.0%	0.0%	22.2%	0.0%	22.2%	14.8%
4	11.1%	11.1%	100.0%	77.8%	100.0%	100.0%	11.1%	100.0%	77.8%	65.4%
5	0.0%	0.0%	0.0%	11.1%	0.0%	0.0%	0.0%	0.0%	0.0%	1.2%
6	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
7	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
8	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
9	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
10	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
Totale	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%

Si può notare dalle tabelle come chi ha imparato a giocare 3 (o 4) continua generalmente a scegliere lo stesso numero. L'effetto è palese per alcuni giocatori, che dall'inizio alla fine dei turni dichiarano sempre 3 o 4.



Punteggio calcolato nel modo"vecchio"

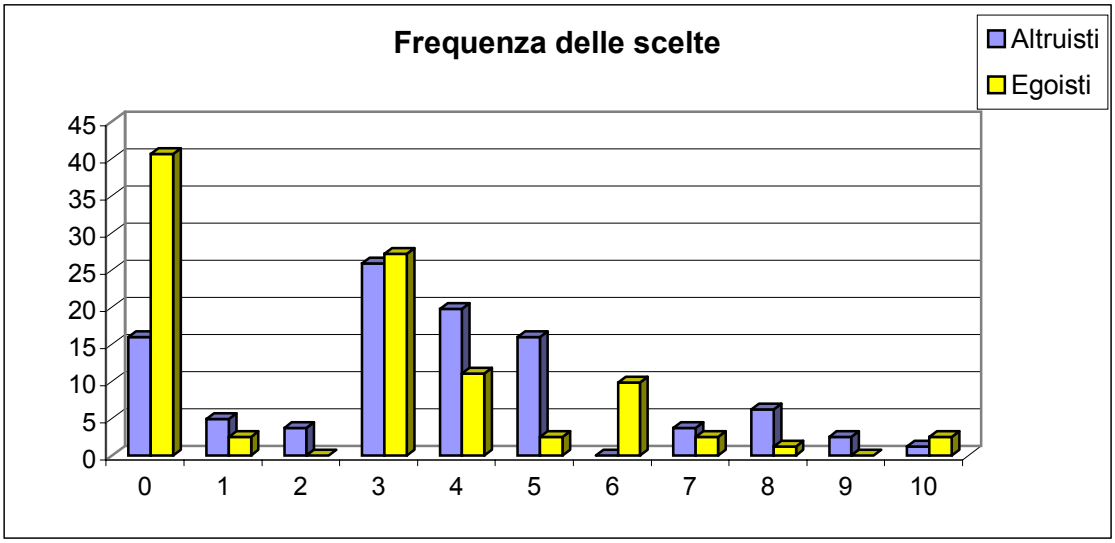
L'effetto training non si registra invece con il punteggio calcolato con il metodo utilizzato già nel primo esperimento. In questo caso si ha invece una fortissima eterogeneità (tra i vari soggetti ma anche per diversi soggetti nel corso del gioco) per entrambi i gruppi di controllo. Inoltre, tra chi ha giocato il primo turno con gli agenti egoisti emerge una quota più alta di 0, mentre per chi ha giocato con gli agenti artificiali altruisti prevale una quota elevata di numeri maggiori o uguali a 4, segno possibile di una volontà di adottare un comportamento di reciprocità con gli altri e di non comportarsi da *free riders*.

Tabella 13

	Training con altruisti									**Totale**
Scelta	2	7	14	15	21	25	29	33	36	
0	44.4%	0.0%	22.2%	0.0%	55.6%	0.0%	22.2%	0.0%	0.0%	16.0%
1	33.3%	0.0%	11.1%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	4.9%
2	11.1%	0.0%	11.1%	0.0%	0.0%	11.1%	0.0%	0.0%	0.0%	3.7%
3	11.1%	11.1%	33.3%	33.3%	22.2%	0.0%	22.2%	100.0%	0.0%	25.9%
4	0.0%	88.9%	11.1%	0.0%	22.2%	22.2%	0.0%	0.0%	33.3%	19.8%
5	0.0%	0.0%	11.1%	55.6%	0.0%	22.2%	55.6%	0.0%	0.0%	16.0%
6	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
7	0.0%	0.0%	0.0%	0.0%	0.0%	11.1%	0.0%	0.0%	22.2%	3.7%
8	0.0%	0.0%	0.0%	0.0%	0.0%	22.2%	0.0%	0.0%	33.3%	6.2%
9	0.0%	0.0%	0.0%	0.0%	0.0%	11.1%	0.0%	0.0%	11.1%	2.5%
10	0.0%	0.0%	0.0%	11.1%	0.0%	0.0%	0.0%	0.0%	0.0%	1.2%
Totale	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%

Tabella 14

	Training con egoisti									**Totale**
Scelta	1	6	10	18	19	20	24	26	35	
0	33.3%	0.0%	0.0%	100.0%	44.4%	88.9%	100.0%	0.0%	0.0%	40.7%
1	11.1%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	11.1%	2.5%
2	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
3	55.6%	66.7%	0.0%	0.0%	11.1%	11.1%	0.0%	100.0%	0.0%	27.2%
4	0.0%	22.2%	11.1%	0.0%	11.1%	0.0%	0.0%	0.0%	55.6%	11.1%
5	0.0%	11.1%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	11.1%	2.5%
6	0.0%	0.0%	88.9%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	9.9%
7	0.0%	0.0%	0.0%	0.0%	11.1%	0.0%	0.0%	0.0%	11.1%	2.5%
8	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	11.1%	1.2%
9	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
10	0.0%	0.0%	0.0%	0.0%	22.2%	0.0%	0.0%	0.0%	0.0%	2.5%
Totale	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%


Frequenza delle scelte
Altruisti
Egoisti
45
40
35
30
25
20
15
10
5
0
0
1
2
3
4
5
6
7
8
9
10

CONCLUSIONE

All'interno di questo lavoro sono stati delineati i tratti fondamentali dell'economia sperimentale e, attraverso l'analisi dei suoi pregi e dei suoi difetti, le potenzialità a disposizione degli sperimentatori. Sono state illustrate le radici della sperimentazione in economia e i filoni di ricerca che si sono sviluppati nel tempo. In particolare abbiamo ricercato gli eventuali comportamenti cooperativi ed altruistici che possono sorgere in un ambiente economico attraverso gli esperimenti svolti presso la Facoltà di Economia. L'idea alla base della nostra sperimentazione nasce dall'intenzione di verificare se il comportamento umano sia influenzabile dall'apprendimento effettuato tramite fasi di interazione con agenti artificiali. Tale stadio di apprendimento con agenti virtuali egoisti (altruisti) guidati da un algoritmo poteva:

- non generare alcun effetto sul comportamento umano nella fase successiva;
- indurre gli esseri umani ad un comportamento altruistico, nato per assecondare il comportamento della squadra;
- portare ad un atteggiamento egoistico per reazione al comportamento della squadra.

Il programma utilizzato ci ha permesso di raggiungere gli obbiettivi che ci eravamo preposti e di verificare le ipotesi, adattandosi alle nostre esigenze.

Swiee consente di utilizzare Swarm per effettuare esperimenti in modo da far interagire due modi diversi di creare, analizzare e verificare teorie economiche:

- "le simulazioni", per mezzo delle quali si cerca di riprodurre parte della realtà che ci circonda per meglio comprenderne il funzionamento;
- "gli esperimenti", in cui un campione di agenti reali viene coinvolto in situazioni predeterminate al fine di studiarne le azioni e reazioni, cercando di comprenderne i ragionamenti e le strategie.

I risultati degli esperimenti ci hanno permesso di constatare che durante la fase di apprendimento con agenti artificiali, gli agenti umani hanno dimostrato una marcata tendenza all'adattamento al gioco della squadra, apprendendo e coordinandosi. Durante la fase di interazione tra i soli umani è però emerso quanto sia stato difficile il coordinamento. Questo può derivare dal fatto che le scelte "suggerite" dagli algoritmi, coerenti con una strategia semplice e precisa, inducevano il giocatore umano ad assecondare il comportamento degli agenti artificiali, mentre in una squadra di soli agenti umani era più difficile individuare le strategie dei compagni e coordinarsi per

massimizzare il risultato proprio e del gruppo. Tra le situazioni estreme si sono riscontrati atteggiamenti puramente egoistici che hanno creato notevoli difficoltà nella ricerca di un equilibrio e che hanno impedito alla squadra di raggiungere un punteggio elevato e casi in cui la mancata comprensione del gioco ha minato qualunque possibilità di raggiungere un qualche equilibrio.

L'analisi dei dati ha mostratoato che si sono delineate strategie di comportamento abbastanza precise: con atteggiamenti altruisti, nei confronti di giocatori che dimostravano di seguire strategie coerenti o cooperative, e egoistici, verso chi adottava strategie di difficile comprensione od eccessivamente egoistiche.

La tendenza generale, riscontrabile nei grafici, mostra infine che la fase di apprendimento influenza in maniera significativa il successivo comportamento del giocatore umano, in entrambi gli esperimenti e nonostante le differenti caratteristiche dei due: la maggior parte dei partecipanti, infatti, ha dimostrato di accettare il comportamento imposto dagli agenti artificiali, proiettando anche nella seconda fase la strategia "suggerita" dal gioco con gli artificiali.

Purtroppo, i "pochi" dati raccolti non ci permettono di dedurre teorie, sarebbe infatti opportuno essere in possesso di un'ampia casistica per poter eliminare eventuali gruppi che hanno mostrato di non aver compreso il gioco e per permetterci di effettuare uno studio più approfondito. Sicuramente gli elementi raccolti ci lasciano affermare che, anche in economia, esistono forme di altruismo e cooperazione e che tali comportamenti sono molto più frequenti di quanto si potesse immaginare. Ulteriori esperimenti, svolti con l'intervento di più partecipanti e anche con l'utilizzo dell'internet permetteranno di confermare quanto osservato nelle sedi di Alessandria e Torino. L'internet consentirebbe, infatti, a un numero molto ampio di giocatori di interagire, anche dalla propria abitazione, dando la possibilità di raccogliere un più elevato numero di dati.

APPENDICE

Listato dei programmi

Esperimento del 3 maggio 2001

StarSum10.java

```
// Game Theory - Model 1 Copyright © 2000
Riccardo Boero
// This library is distributed without any
warranty; without even the
// implied warranty of merchantability or
fitness for a particular purpose.
// See file LICENSE for details and terms of
copying.

import swarm.Globals;

public class StartSum10 {
    // Main function
    public static void main (String[] args) {
        Globals.env.initSwarm ("sum10",
"2.1.1", "bug-swarm@santafe.edu", args);

 Sum10ObserverSwarm topLevelSwarm =
     new Sum10ObserverSwarm
(Globals.env.globalZone);
 Globals.env.setWindowGeometryRecordName
(topLevelSwarm, "topLevelSwarm");

 topLevelSwarm.buildObjects ();
 topLevelSwarm.buildActions ();
 topLevelSwarm.activateIn (null);
 topLevelSwarm.go ();
 topLevelSwarm.drop ();
    }
}
```

Player.java

```
import swarm.Globals;
import java.io.*;

// The virtual representation of the player

public class Player {

    // paramethers
    public int points, choice, choice2,
payoffArray[], id, cumPoints, accInstructions;
```

```
    public int id1, id2, sum3, id3, id4;
    public Player playerArray[];
    public Timer timer;
    public PrintWriter buff;
    public String data;
    public RuleMaster ruleMaster;
    public int numPlayers, changed;

    // constructor
    public Player (int b, int a[], int c,
Player[] pl, Timer t, RuleMaster r) {
 choice = 0;
 points = 0;
 cumPoints = 0;
 accInstructions = 0;
 numPlayers = b;
 payoffArray = a;
 id = c;
 playerArray = pl;
 timer = t;
 ruleMaster = r;
 buff = timer.getBuffer();
 changed = 0;
 int j;
 //choice2 = 1;
      if (id == 0 || id == 3 || id == 6 || id
== 9 || id == 12 || id == 15 || id == 18 || id
== 21 || id == 24 || id == 27 || id == 30 || id
== 33 || id == 36 || id == 39 || id == 42 || id
== 45 || id == 48 || id == 51 || id == 54 || id
== 57 || id == 60 || id == 63 || id == 66 || id
== 69 || id == 72 || id == 75 || id == 78 || id
== 81 || id == 84 || id == 87 || id == 90 || id
== 93 || id == 96 || id == 99 || id == 102 ||
id == 105 || id == 108 || id == 111 || id ==
114 || id == 117 || id == 120 || id == 123) {
     id1 = id + 1;
     id2 = id + 2;
 }
       if (id == 1 || id == 4 || id == 7 || id
== 10 || id == 13 || id == 16 || id == 19 || id
== 22 || id == 25 || id == 28 || id == 31 || id
== 34 || id == 37 || id == 40 || id == 43 || id
== 46 || id == 49 || id == 52 || id == 55 || id
== 58 || id == 61 || id == 64 || id == 67 || id
== 70 || id == 73 || id == 76 || id == 79 || id
== 82 || id == 85 || id == 88 || id == 91 || id
== 94 || id == 97 || id == 100 || id == 103 ||
id == 106 || id == 109 || id == 112 || id ==
115 || id == 118 || id == 121 || id == 124) {
     id1 = id - 1;
     id2 = id + 1;
 }
 if (id == 2 || id == 5 || id == 8 || id == 11
|| id == 14 || id == 17 || id == 20 || id == 23
|| id == 26 || id == 29 || id == 32 || id == 35
|| id == 38 || id == 41 || id == 44 || id == 47
|| id == 50 || id == 53 || id == 56 || id == 59
|| id == 62 || id == 65 || id == 68 || id == 71
```

```
|| id == 74 || id == 77 || id == 80 || id == 83
|| id == 86 || id == 89 || id == 92 || id == 95
|| id == 98 || id == 101 || id == 104 || id ==
107 || id == 110 || id == 113 || id == 116 ||
id == 119 || id == 122 || id == 125) {
      id1 = id - 2;
      id2 = id - 1;
  }
  if ((id >= 0 && id <=2) || (id >= 9 && id
<=11) || (id >= 18 && id <=20) || (id >= 27 &&
id <=29) || (id >= 36 && id <=38) || (id >= 45
&& id <=47) || (id >= 54 && id <=56) || (id >=
63 && id <=65) || (id >= 72 && id <=74) || (id
>= 81 && id <=83) || (id >= 90 && id <=92) ||
(id >= 99 && id <=101) || (id >= 108 && id
<=110) || (id >= 117 && id <=119)) {
      id3 = id + 3;
      id4 = id + 6;
  }
  if ((id >= 3 && id <=5) || (id >= 12 && id
<=14) || (id >= 21 && id <=23) || (id >= 30 &&
id <=32) || (id >= 39 && id <=41) || (id >= 48
&& id <=50) || (id >= 57 && id <=59) || (id >=
66 && id <=68) || (id >= 75 && id <=77) || (id
>= 84 && id <=86) || (id >= 93 && id <=95) ||
(id >= 102 && id <=104) || (id >= 111 && id
<=113) || (id >= 120 && id <=122)) {
      id3 = id - 3;
      id4 = id + 3;
  }
  if ((id >= 6 && id <=8) || (id >= 15 && id
<=17) || (id >= 24 && id <=26) || (id >= 33 &&
id <=35) || (id >= 42 && id <=44) || (id >= 51
&& id <=53) || (id >= 60 && id <=62) || (id >=
69 && id <=71) || (id >= 78 && id <=80) || (id
>= 87 && id <=89) || (id >= 96 && id <=98) ||
(id >= 105 && id <=107) || (id >= 114 && id
<=116) || (id >= 123 && id <=125)) {
      id3 = id - 6;
      id4 = id - 3;
  }
    }

    // methods
    public int getPoints () {
  return points;
    }

    public int[] getOtherChoice () {
  int i[];
  i = new int[2];
  i[0] = playerArray[id1].getChoice();
  i[1] = playerArray[id2].getChoice();
  return i;
    }

    public Object setChoice (int i) {
  choice = i;
  timer.setBlock(id);
```

```
return this;
   }

   public int getChoice () {
return choice;
   }

   public int getChoice2 () {
return choice2;
   }

   public Object setChoice2 () {
choice2 = choice;
return this;
   }

   public int getCumPoints () {
return cumPoints;
   }

   public Object setInstructions () {
accInstructions++;
return this;
   }

   public Object nullInstructions () {
accInstructions = 0;
return this;
   }

   public int getSum3 () {
int sum3b, i[];
i = getOtherChoice();
sum3b = choice + i[0] + i[1];
return sum3b;
   }

   // Schedule's method
   public Object step () {
     if (id == 0 || id == 3 || id == 6 || id
== 9 || id == 12 || id == 15 || id == 18 || id
== 21 || id == 24 || id == 27 || id == 30 || id
== 33 || id == 36 || id == 39 || id == 42 || id
== 45 || id == 48 || id == 51 || id == 54 || id
== 57 || id == 60 || id == 63 || id == 66 || id
== 69 || id == 72 || id == 75 || id == 78 || id
== 81 || id == 84 || id == 87 || id == 90 || id
== 93 || id == 96 || id == 99 || id == 102 ||
id == 105 || id == 108 || id == 111 || id ==
114 || id == 117 || id == 120 || id == 123) {
 }
     else {
setChoice(ruleMaster.getChoice(this));
     }
return this;
   }

   public Object step2 () {
while (!(timer.askBlock())) {
```

```
        continue;
    }


    //Upgrade payoffs
    //int otherChoice[];
    //otherChoice = getOtherChoice ();

    sum3 = getSum3();
    if (sum3 == 10)
        points = 40 - choice;
    if (sum3 > 10)
        points = 30 - choice;
    if (sum3 < 10)
        points = -choice;

    //if (otherChoice == expectation)
    //  payoff ++;
    if (changed == 1) {
        changed = 0;
        cumPoints = 0;
    }
    if (timer.askChange()) {
       changed = 1;
       id1 = id3;
       id2 = id4;
    }
    cumPoints += points;
    timer.setBlock2();
    setChoice2();
    data = (String) (id + " " + choice + " " +
points + " " + cumPoints + " " + sum3);
    //System.out.println(data);
    buff.println(data);
    return this;
       }
}
```

RemoteServer.java

```
import java.rmi.Naming;
import java.rmi.*;
import java.net.*;
import java.io.*;
import java.rmi.server.*;
import java.rmi.registry.LocateRegistry;
// Remote Server's class easily manages in
Swarm simulation
// RMI Java technology
```

```
public class RemoteServer {
    Player[] playerArray;
    Timer timer;
    private static RMIRegistry  rmi;
    public RemoteServer(Player[] pl, Timer t,
int[] payoffArray) {
 playerArray = pl;
 timer = t;
 try {
     activateRegistry();
     //System.setSecurityManager(new
MySecurityManager());
     RemoteSum10 c = new
RemoteSum10Impl(playerArray, timer,
payoffArray);
     // Pay attention to the IP of your host
and to the port:
     // default port of RMIRegistry is 1099

Naming.rebind("//130.192.140.49:1098/RemoteSum1
0", c);
     System.out.println("RemoteServer is
registered.");
 } catch (Exception e) {
     System.out.println("Trouble in get
RemoteSum10: " + e);
 }
    }
    public void activateRegistry() {

 try
     {
       rmi = new RMIRegistry();
     }
 catch ( java.rmi.UnknownHostException uhe )
     {
       System.out.println( "The host computer
name you have specified, does not match your
real computer name." );
     }
 catch ( RemoteException re )
     {
       System.out.println( "Error starting
service" );
       System.out.println( "" + re );
     }
 catch ( MalformedURLException mURLe )
     {
       System.out.println( "Internal error" +
mURLe );
     }
 catch ( NotBoundException nbe )
     {
       System.out.println( "Not Bound" );
       System.out.println( "" + nbe );
     }
    }
}
```

RemoteSum10.java

```
import java.rmi.*;

// Interface to Server Methods

public interface RemoteSum10 extends Remote {
    public void setChoice(int i, int v) throws
RemoteException;
    public int getCumPoints(int i) throws
RemoteException;
    public boolean askRound(int i) throws
RemoteException;
    public boolean askBlock2() throws
RemoteException;
    public int getPoints(int i) throws
RemoteException;
    public int[] getPayoffArray() throws
RemoteException;
    public int getChoice(int i) throws
RemoteException;
    public boolean askEnd() throws
RemoteException;
    public int askNumRound() throws
RemoteException;
    public void setBlock3() throws
RemoteException;
    public boolean askChange() throws
RemoteException;
}
```

RemoteSum10Impl.java

```
import java.rmi.*;
import java.rmi.server.*;
import java.util.*;

// Implementation of Server's Methods.

public class RemoteSum10Impl extends
UnicastRemoteObject
    implements RemoteSum10 {
    public Player[] playerArray;
    public Timer timer;
    public int[] payoffArray;
    public RemoteSum10Impl(Player[] pl, Timer
t, int[] p) throws RemoteException {
```

```
  super();
  playerArray = pl;
  timer = t;
  payoffArray = p;
     }
     public void setChoice(int i, int v) throws
RemoteException {
  int id = i, value = v;
  playerArray[id].setChoice(value);
     }
     public int getCumPoints(int i) throws
RemoteException  {
  int id = i;
  int value = playerArray[id].getCumPoints();
  return value;
  }
     public boolean askRound(int i) throws
RemoteException  {
  int id = i;
  boolean ok = timer.getRound(id);
  return ok;
     }
     public void setBlock3() throws
RemoteException  {
  timer.setBlock3();
     }
     public boolean askBlock2() throws
RemoteException  {
  boolean ok = (timer.askBlock2());
  return ok;
  }
     public int getPoints(int i) throws
RemoteException  {
  int id = i;
  int value = playerArray[id].getPoints();
  return value;
     }
     public int[] getPayoffArray() throws
RemoteException  {
  return payoffArray;
     }

     public int getChoice(int i) throws
RemoteException  {
  int value = playerArray[i].getChoice2();
  return value;
     }
     public boolean askEnd() throws
RemoteException  {
  boolean ok;
  if (timer.cycles >= (timer.numCycles - 1))
      ok = true;
  else
      ok = false;
  return ok;
     }
     public boolean askChange() throws
RemoteException  {
  boolean ok;
```

```
 if (timer.askChange())
     ok = true;
 else
     ok = false;
 return ok;
    }
    public int askNumRound() throws
RemoteException  {
 int i = timer.cycles;
 return i;
    }
}
```

RmiRegistry.java

```
import java.rmi.Naming;
import java.rmi.*;
import java.net.*;
import java.io.*;
import java.rmi.server.*;
import java.rmi.registry.LocateRegistry;

public class RMIRegistry {
    public RMIRegistry() throws
RemoteException, MalformedURLException,
NotBoundException {
 LocateRegistry.createRegistry( 1098 );
 System.out.println( "Registry created on host
computer on port 1098.");
    }
}
```

RuleMaster.java

```
import swarm.Globals;
//import java.io.*;

// Rule Master

public class RuleMaster {

    // paramethers
    public Timer timer;
    public int numPlayers;
    // constructor
    public RuleMaster (Timer t, int r) {
      timer = t;
      numPlayers = r;
 //int i, j, z, x;
```

```
    }

    // methods
    public int getChoice (Player a) {
      int b = 0;
      int j;

      /*if (a.id == 0 || a.id == 3 || a.id == 6
|| a.id == 9 || a.id == 12 || a.id == 15
         || a.id == 18 || a.id == 21 || a.id ==
24 || a.id == 27 || a.id == 30 || a.id == 33
         || a.id == 36 || a.id == 39 || a.id ==
42 || a.id == 45 || a.id == 48 || a.id == 51) {
        if (timer.cycles <= 1) {
            b = 3;
        }
        else {
               int temp =
Globals.env.uniformIntRand.getIntegerWithMin$wi
thMax (0, 4);
            if (temp == 0)
             b = 0;
            if (temp == 1)
             b = 1;
            if (temp == 2)
             b = 3;
             if (temp == 3)
             b = 4;
            if (temp == 4)
             b = 10;
        }
      }*/

// agenti altruisti

      if (a.id == 4 || a.id == 7 || a.id == 13
|| a.id == 19 || a.id == 25 || a.id == 31 ||
a.id == 37 || a.id == 43 || a.id == 49 || a.id
== 55 || a.id == 61 || a.id == 67 || a.id == 73
|| a.id == 79 || a.id == 85 || a.id == 91 ||
a.id == 97 || a.id == 103 || a.id == 109 ||
a.id == 115 || a.id == 121) {
            b = 4;
      }

      if (a.id == 5 || a.id == 8 || a.id == 14
|| a.id == 20 || a.id == 26 || a.id == 32 ||
a.id == 38 || a.id == 38 || a.id == 44 || a.id
== 50 || a.id == 56 || a.id == 62 || a.id == 68
|| a.id == 74 || a.id == 80 || a.id == 86 ||
a.id == 92 || a.id == 98 || a.id == 104 || a.id
== 110 || a.id == 116 || a.id == 122) {
        if (timer.cycles == 0) {
          b = 3;
        }
        else {
            if ((a.playerArray[a.id1].choice2 +
a.playerArray[a.id2].choice2) < 10) {
```

```
                if
((a.playerArray[a.id1].choice2 +
a.playerArray[a.id2].choice2 + 1) < 10){
                  if
((a.playerArray[a.id1].choice2 +
a.playerArray[a.id2].choice2 + 3) < 10){
                    if
((a.playerArray[a.id1].choice2 +
a.playerArray[a.id2].choice2 + 4) < 10){
                      b = 10;
                    }
                    else
                      b = 4;
                  }
                  else
                    b = 3;
                }
                else
                  b = 1;
            }
            else
               b = 0;

        }
      }

// agenti egoisti

      if (a.id == 1 || a.id == 10 || a.id == 16
|| a.id == 22 || a.id == 28 || a.id == 34 ||
a.id == 40 || a.id == 46 || a.id == 52 || a.id
== 58 || a.id == 64 || a.id == 70 || a.id == 76
|| a.id == 82 || a.id == 88 || a.id == 94 ||
a.id == 100 || a.id == 106 || a.id == 112 ||
a.id == 118 || a.id == 124) {
            b = 3;
      }

      if (a.id == 2 || a.id == 11 || a.id == 17
|| a.id == 23 || a.id == 29 || a.id == 35 ||
a.id == 41 || a.id == 47 || a.id == 53 || a.id
== 59 || a.id == 65 || a.id == 71 || a.id == 77
|| a.id == 83 || a.id == 89 || a.id == 95 ||
a.id == 101 || a.id == 107 || a.id == 113 ||
a.id == 119 || a.id == 125) {
        if (timer.cycles == 0) {
          b = 3;
        }
        else {
            if (((a.playerArray[a.id1].choice2
+ a.playerArray[a.id2].choice2) < 7 ||
(a.playerArray[a.id1].choice2 +
a.playerArray[a.id2].choice2) > 10)){
                  b = 0;
            }
            else
               b = 3;
        }
      }
```

```
  return b;
     }
}
```

Sum10Client.java

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
import java.net.*;
import java.rmi.*;
import java.rmi.Naming;

public class Sum10Client extends Applet  {
    int value, value2, id, id1, id2, id3, id4,
payoffArray[];
    String identification, valueS;
    boolean updated, activated;
    //Object parentFrame;
    RemoteSum10 server;

    java.awt.Label label1 = new
java.awt.Label();
    java.awt.Checkbox radioButton1 = new
java.awt.Checkbox();
    java.awt.CheckboxGroup Group1 = new
java.awt.CheckboxGroup();
    java.awt.Checkbox radioButton2 = new
java.awt.Checkbox();
    java.awt.Checkbox radioButton3 = new
java.awt.Checkbox();
    java.awt.Checkbox radioButton4 = new
java.awt.Checkbox();
    java.awt.Checkbox radioButton5 = new
java.awt.Checkbox();
    java.awt.Label label2 = new
java.awt.Label();
    java.awt.Label label11 = new
java.awt.Label();
    java.awt.Label label12 = new
java.awt.Label();
    java.awt.Label label13 = new
java.awt.Label();
    java.awt.Label label14 = new
java.awt.Label();
    java.awt.TextField textField7 = new
java.awt.TextField();
    java.awt.TextField textField8 = new
java.awt.TextField();
    java.awt.TextField textField9 = new
java.awt.TextField();
    java.awt.TextField textField10 = new
java.awt.TextField();
```

```
    java.awt.TextField textField11 = new
java.awt.TextField();
    java.awt.TextField textField12 = new
java.awt.TextField();
    java.awt.TextField textField13 = new
java.awt.TextField();
    java.awt.Label label3 = new
java.awt.Label();
    java.awt.Button button1 = new
java.awt.Button();
    java.awt.Label label4 = new
java.awt.Label();
    java.awt.TextField textField1 = new
java.awt.TextField();
    java.awt.Label label5 = new
java.awt.Label();
    java.awt.TextField textField2 = new
java.awt.TextField();
    java.awt.Label label6 = new
java.awt.Label();
    java.awt.TextField textField3 = new
java.awt.TextField();
    java.awt.Label label7 = new
java.awt.Label();
    java.awt.TextField textField4 = new
java.awt.TextField();
    java.awt.Label label8 = new
java.awt.Label();
    java.awt.Label label9 = new
java.awt.Label();
    java.awt.TextField textField5 = new
java.awt.TextField();
    java.awt.Label label15 = new
java.awt.Label();
    java.awt.Label label16 = new
java.awt.Label();
    java.awt.Label label10 = new
java.awt.Label();
    java.awt.TextField textField6 = new
java.awt.TextField();
    java.awt.TextArea textArea1 = new
java.awt.TextArea();
    java.awt.Button button2 = new
java.awt.Button();
    //aggiunte
    java.awt.Label label17 = new
java.awt.Label();
    java.awt.TextField textField14 = new
java.awt.TextField();

    public void init() {
  identification =
getParameter("identification");
  id = Integer.parseInt(identification);
  setLayout(null);
  setSize(364,398);
  activated = false;
  label1.setText("Scelta:");
  add(label1);
```

```
 label1.setFont(new Font("Dialog", Font.BOLD,
12));
 label1.setBounds(12,12,50,12);
 radioButton1.setCheckboxGroup(Group1);
 radioButton1.setLabel("A");
 add(radioButton1);
 radioButton1.setFont(new Font("Dialog",
Font.BOLD, 12));
 radioButton1.setBounds(70,12,28,12);
 radioButton2.setCheckboxGroup(Group1);
 radioButton2.setLabel("B");
 add(radioButton2);
 radioButton2.setFont(new Font("Dialog",
Font.BOLD, 12));
 radioButton2.setBounds(126,12,28,12);
 radioButton3.setCheckboxGroup(Group1);
 radioButton3.setLabel("C");
 add(radioButton3);
 radioButton3.setFont(new Font("Dialog",
Font.BOLD, 12));
 radioButton3.setBounds(182,12,28,12);
 radioButton4.setCheckboxGroup(Group1);
 radioButton4.setLabel("D");
 add(radioButton4);
 radioButton4.setFont(new Font("Dialog",
Font.BOLD, 12));
 radioButton4.setBounds(238,12,28,12);
 radioButton5.setCheckboxGroup(Group1);
 radioButton5.setLabel("E");
 add(radioButton5);
 radioButton5.setFont(new Font("Dialog",
Font.BOLD, 12));
 radioButton5.setBounds(294,12,28,12);
 label2.setText("A:");
 label2.setAlignment(java.awt.Label.RIGHT);
 add(label2);
 label2.setBounds(50,36,20,24);
 add(textField7);
 textField7.setEditable(false);
 textField7.setBounds(70,36,30,24);
 label11.setText("B:");
 label11.setAlignment(java.awt.Label.RIGHT);
 add(label11);
 label11.setBounds(106,36,20,24);
 add(textField8);
 textField8.setEditable(false);
 textField8.setBounds(126,36,30,24);
 label12.setText("C:");
 label12.setAlignment(java.awt.Label.RIGHT);
 add(label12);
 label12.setBounds(162,36,20,24);
 add(textField9);
 textField9.setEditable(false);
 textField9.setBounds(182,36,30,24);
 label13.setText("D:");
 label13.setAlignment(java.awt.Label.RIGHT);
 add(label13);
 label13.setBounds(218,36,20,24);
 add(textField10);
```

```
  textField10.setEditable(false);
  textField10.setBounds(238,36,30,24);
  label14.setText("E:");
  label14.setAlignment(java.awt.Label.RIGHT);
  add(label14);
  label14.setBounds(274,36,20,24);
  add(textField11);
  textField11.setEditable(false);
  textField11.setBounds(294,36,30,24);
  //modificate
  label15.setText("Somma della tua terna:");
  label15.setAlignment(java.awt.Label.RIGHT);
  add(label15);
  label15.setBounds(12,218,130,24);
  add(textField12);
  textField12.setEditable(false);
  textField12.setBounds(150,218,40,24);
  label16.setText("Somma tuoi punti:");
  label16.setAlignment(java.awt.Label.RIGHT);
  add(label16);
  label16.setBounds(200,218,100,24);
  add(textField13);
  textField13.setEditable(false);
  textField13.setBounds(310,218,40,24);
  //fine
  //aggiunte
  label17.setText("Turno n.");
  label17.setAlignment(java.awt.Label.RIGHT);
  add(label17);
  label17.setBounds(120,184,50,24);
  add(textField14);
  textField14.setEditable(false);
  textField14.setBounds(170,184,40,24);

  label3.setText("Risultati dell\'ultimo
incontro:");
  add(label3);
  label3.setFont(new Font("Dialog",
Font.BOLD|Font.ITALIC, 12));
  label3.setBounds(12,96,180,24);
  button1.setLabel("Invia i dati");
  add(button1);
  button1.setBackground(java.awt.Color.lightGray
);
  button1.setBounds(12,70,340,24);
  button1.addActionListener(new
ButtonHandler());
  label4.setText("Tua scelta:");
  label4.setAlignment(java.awt.Label.RIGHT);
  add(label4);
  label4.setBounds(12,120,83,24);
  add(textField1);
  textField1.setEditable(false);
  textField1.setBounds(95,120,40,24);
  label5.setText("Scelta di 1:");
  label5.setAlignment(java.awt.Label.RIGHT);
  add(label5);
  label5.setBounds(135,120,65,24);
  add(textField2);
```

```
  textField2.setEditable(false);
  textField2.setBounds(200,120,40,24);
  label6.setText("Scelta di 2:");
  label6.setAlignment(java.awt.Label.RIGHT);
  add(label6);
  label6.setBounds(240,120,72,24);
  add(textField3);
  textField3.setEditable(false);
  textField3.setBounds(312,120,40,24);

  label7.setText("Tuo punteggio:");
  label7.setAlignment(java.awt.Label.RIGHT);
  add(label7);
  label7.setBounds(12,156,83,24);
  add(textField4);
  textField4.setEditable(false);
  textField4.setBounds(95,156,40,24);
  label9.setText("Punti di 1:");
  label9.setAlignment(java.awt.Label.RIGHT);
  add(label9);
  label9.setBounds(135,156,65,24);
  add(textField5);
  textField5.setEditable(false);
  textField5.setBounds(200,156,40,24);
  label10.setText("Punti di 2:");
  label10.setAlignment(java.awt.Label.RIGHT);
  add(label10);
  label10.setBounds(240,156,72,24);
  add(textField6);
  textField6.setEditable(false);
  textField6.setBounds(312,156,40,24);

  button2.setLabel("Attiva");
  add(button2);
  button2.setBackground(java.awt.Color.lightGray
);
  button2.setBounds(12,244,340,24);
  button2.addActionListener(new
ButtonHandler());

  label8.setText("Messaggi:");
  add(label8);
  label8.setFont(new Font("Dialog",
Font.BOLD|Font.ITALIC, 12));
  label8.setBounds(12,268,132,24);
  add(textArea1);
  textArea1.setBounds(12,292,340,84);

  updated = true;


     if (id == 0 || id == 3 || id == 6 || id ==
9 || id == 12 || id == 15 || id == 18 || id ==
21 || id == 24 || id == 27 || id == 30 || id ==
33 || id == 36 || id == 39 || id == 42 || id ==
45 || id == 48 || id == 51 || id == 54 || id ==
57 || id == 60 || id == 63 || id == 66 || id ==
69 || id == 72 || id == 75 || id == 78 || id ==
81 || id == 84 || id == 87 || id == 90 || id ==
```

```
93 || id == 96 || id == 99 || id == 102 || id
== 105 || id == 108 || id == 111 || id == 114
|| id == 117 || id == 120 || id == 123) {
      id1 = id + 1;
      id2 = id + 2;
  }
        if (id == 1 || id == 4 || id == 7 || id
== 10 || id == 13 || id == 16 || id == 19 || id
== 22 || id == 25 || id == 28 || id == 31 || id
== 34 || id == 37 || id == 40 || id == 43 || id
== 46 || id == 49 || id == 52 || id == 55 || id
== 58 || id == 61 || id == 64 || id == 67 || id
== 70 || id == 73 || id == 76 || id == 79 || id
== 82 || id == 85 || id == 88 || id == 91 || id
== 94 || id == 97 || id == 100 || id == 103 ||
id == 106 || id == 109 || id == 112 || id ==
115 || id == 118 || id == 121 || id == 124) {
      id1 = id - 1;
      id2 = id + 1;
  }
  if (id == 2 || id == 5 || id == 8 || id == 11
|| id == 14 || id == 17 || id == 20 || id == 23
|| id == 26 || id == 29 || id == 32 || id == 35
|| id == 38 || id == 41 || id == 44 || id == 47
|| id == 50 || id == 53 || id == 56 || id == 59
|| id == 62 || id == 65 || id == 68 || id == 71
|| id == 74 || id == 77 || id == 80 || id == 83
|| id == 86 || id == 89 || id == 92 || id == 95
|| id == 98 || id == 101 || id == 104 || id ==
107 || id == 110 || id == 113 || id == 116 ||
id == 119 || id == 122 || id == 125) {
      id1 = id - 2;
      id2 = id - 1;
  }
  if ((id >= 0 && id <=2) || (id >= 9 && id
<=11) || (id >= 18 && id <=20) || (id >= 27 &&
id <=29) || (id >= 36 && id <=38) || (id >= 45
&& id <=47) || (id >= 54 && id <=56) || (id >=
63 && id <=65) || (id >= 72 && id <=74) || (id
>= 81 && id <=83) || (id >= 90 && id <=92) ||
(id >= 99 && id <=101) || (id >= 108 && id
<=110) || (id >= 117 && id <=119)) {
      id3 = id + 3;
      id4 = id + 6;
  }
  if ((id >= 3 && id <=5) || (id >= 12 && id
<=14) || (id >= 21 && id <=23) || (id >= 30 &&
id <=32) || (id >= 39 && id <=41) || (id >= 48
&& id <=50) || (id >= 57 && id <=59) || (id >=
66 && id <=68) || (id >= 75 && id <=77) || (id
>= 84 && id <=86) || (id >= 93 && id <=95) ||
(id >= 102 && id <=104) || (id >= 111 && id
<=113) || (id >= 120 && id <=122)) {
      id3 = id - 3;
      id4 = id + 3;
  }
  if ((id >= 6 && id <=8) || (id >= 15 && id
<=17) || (id >= 24 && id <=26) || (id >= 33 &&
id <=35) || (id >= 42 && id <=44) || (id >= 51
```

```
&& id <=53) || (id >= 60 && id <=62) || (id >=
69 && id <=71) || (id >= 78 && id <=80) || (id
>= 87 && id <=89) || (id >= 96 && id <=98) ||
(id >= 105 && id <=107) || (id >= 114 && id
<=116) || (id >= 123 && id <=125)) {
      id3 = id - 6;
      id4 = id - 3;
  }
     }
    public boolean askRound(int i){
  boolean aa = false;
  try  {
      aa = server.askRound(i);
  } catch (Exception ex) {
      textArea1.setText(ex.toString());
  }
  return aa;
  }
    public boolean askBlock2(){
  boolean aa = true;
  try  {
      aa = server.askBlock2();
  } catch (Exception ex) {
      textArea1.setText(ex.toString());
  }
  return aa;
    }

    public void update () {
  try  {
                    //Update information

textField1.setText(Integer.toString(value));
      int newChoice1 = server.getChoice(id1);

textField2.setText(Integer.toString(newChoice1)
);
      int newChoice2 = server.getChoice(id2);

textField3.setText(Integer.toString(newChoice2)
);
      int newPoints = server.getPoints(id);

textField4.setText(Integer.toString(newPoints))
;
      int newPoints1 = server.getPoints(id1);

textField5.setText(Integer.toString(newPoints1)
);
      int newPoints2 = server.getPoints(id2);

textField6.setText(Integer.toString(newPoints2)
);

textField12.setText(Integer.toString(value+newC
hoice1+newChoice2));
      int newCumPoints =
server.getCumPoints(id);
```

```
textField13.setText(Integer.toString(newCumPoin
ts));
                    //aggiunte
     int newNumRound = server.askNumRound();

textField14.setText(Integer.toString(newNumRoun
d + 1));

     updated = true;
     if (server.askEnd())
       textArea1.setText(" -- L'ESPERIMENTO E'
TERMINATO!!! --\n -- L'ESPERIMENTO E'
TERMINATO!!! --\n -- L'ESPERIMENTO E'
TERMINATO!!! --");
     else {
       if (server.askChange()) {
           textArea1.setText(" -- ATTENZIONE!
Da ora affronti nuovi giocatori! --\n --
ATTENZIONE! Da ora affronti nuovi giocatori! --
\n -- ATTENZIONE! Da ora affronti nuovi
giocatori! --");
           id1 = id3;
           id2 = id4;
       }else
           textArea1.setText("Risultati
aggiornati, ora puoi effettuare una nuova
scelta.");
     }
  } catch (Exception ex) {
     textArea1.setText(ex.toString());
  }
    }

    public void activate () {
  try {
     URL hostURL = getCodeBase();
     String host = hostURL.getHost();
     // Change the port number if necessary
     server =
(RemoteSum10)Naming.lookup("//"+host+":1098/Rem
oteSum10");
     } catch (Exception ex) {
     textArea1.setText(ex.toString());
     }
  try  {
     payoffArray = server.getPayoffArray();
  } catch (Exception ex) {
     textArea1.setText(ex.toString());
  }

  // A message to tell players that the first
round is an example
  //textArea1.setText("ATTENZIONE:\nil primo
turno e' di prova!!!");
  textField7.setText(Integer.toString(payoffArra
y[0]));
  textField8.setText(Integer.toString(payoffArra
y[1]));
```

```
 textField9.setText(Integer.toString(payoffArra
y[2]));
 textField10.setText(Integer.toString(payoffArr
ay[3]));
 textField11.setText(Integer.toString(payoffArr
ay[4]));
 activated = true;
    }

    class ButtonHandler implements
ActionListener {
 public void actionPerformed(ActionEvent ev) {
     String s=ev.getActionCommand();

     if ("Invia i dati".equals(s)) {
       if (Group1.getSelectedCheckbox() ==
null) {
           textArea1.setText("Occorre
selezionare la propria scelta.");
       }
       else {
           if (!(askRound(id))) {
             if (updated) {
                 try  {
                   //Submit values
                   valueS =
Group1.getSelectedCheckbox().getLabel();
                   if (valueS == "A")
                       value = payoffArray[0];
                   if (valueS == "B")
                       value = payoffArray[1];
                   if (valueS == "C")
                       value = payoffArray[2];
                   if (valueS == "D")
                       value = payoffArray[3];
                   if (valueS == "E")
                       value = payoffArray[4];
                   server.setChoice(id, value);
                   updated = false;
                   textArea1.setText("Ok,
adesso puoi provare ad aggiornare i
risultati.");
                 } catch (Exception ex) {

 textArea1.setText(ex.toString());
                 }
             }
             else {
                 textArea1.setText("Prima devi
aggiornare i risultati!!!");
             }
           }
           else {
             textArea1.setText("Prima devi
aggiornare i risultati!!!");
           }
       }
     }
```

```
      if ("Attiva".equals(s)) {
        activate();
        textArea1.setText("Ora puoi iniziare a
giocare.");
        button2.setLabel("Aggiorna i
risultati");
      }
      if ("Aggiorna i risultati".equals(s)) {
        if ((askRound(id))) {
            if ((askBlock2())) {
              if (!(updated)) {
                  try  {
                    //Update information

 textField1.setText(Integer.toString(value));
                    int newChoice1 =
server.getChoice(id1);

 textField2.setText(Integer.toString(newChoice1
));
                    int newChoice2 =
server.getChoice(id2);

 textField3.setText(Integer.toString(newChoice2
));
                    int newPoints =
server.getPoints(id);

 textField4.setText(Integer.toString(newPoints)
);
                    int newPoints1 =
server.getPoints(id1);

 textField5.setText(Integer.toString(newPoints1
));
                    int newPoints2 =
server.getPoints(id2);

 textField6.setText(Integer.toString(newPoints2
));

 textField12.setText(Integer.toString(value+new
Choice1+newChoice2));
                    int newCumPoints =
server.getCumPoints(id);

 textField13.setText(Integer.toString(newCumPoi
nts));
                    //aggiunte
                    int newNumRound =
server.askNumRound() + 1;

 textField14.setText(Integer.toString(newNumRou
nd));

                    updated = true;
                    if (server.askEnd())
```

```
                        textArea1.setText(" --
L'ESPERIMENTO E' TERMINATO!!! --\n --
L'ESPERIMENTO E' TERMINATO!!! --\n --
L'ESPERIMENTO E' TERMINATO!!! --");
                    else
                        {
                          if
(server.askChange()) {

textArea1.setText(" -- ATTENZIONE! Da ora
affronti nuovi giocatori! --\n -- ATTENZIONE!
Da ora affronti nuovi giocatori! --\n --
ATTENZIONE! Da ora affronti nuovi giocatori! --
");
                              id1 = id3;
                              id2 = id4;
                          }
                          else

textArea1.setText("Risultati aggiornati, ora
puoi effettuare una nuova scelta.");
                        }
                    try  {
                        server.setBlock3();
                    } catch (Exception ex) {

textArea1.setText(ex.toString());
                    }
                  } catch (Exception ex) {

 textArea1.setText(ex.toString());
                  }
              }
              else {
                  textArea1.setText("Risultati
gia\' aggiornati.");
              }
            }
            else {
              textArea1.setText("Perfavore
aspetta la scelta degli altri\ne quindi invia
nuovamente la tua scelta.");
            }
        }
        else {
            textArea1.setText("Perfavore aspetta
alcuni secondi la scelta\ndegli altri giocatori
e riprova\nad aggiornare i risultati");
        }

      }
  }
    }
}
```

Sum10ModelSwarm.java

```
import swarm.Globals;
import swarm.Selector;
import swarm.defobj.Zone;
import swarm.defobj.SymbolImpl;

import swarm.activity.Activity;
import swarm.activity.ActionGroup;
import swarm.activity.ActionGroupImpl;
import swarm.activity.Schedule;
import swarm.activity.ScheduleImpl;
import swarm.activity.ActionForEach;

import swarm.collections.List;
import swarm.collections.ListImpl;

import swarm.objectbase.Swarm;
import swarm.objectbase.SwarmImpl;
import swarm.objectbase.VarProbe;
import swarm.objectbase.MessageProbe;
import swarm.objectbase.EmptyProbeMapImpl;

import java.io.*;

/** The Sum10ModelSwarm encapsulates all the
objects used in the
  simulated world itself (but not the user
interface objects) */

public class Sum10ModelSwarm extends SwarmImpl
{

    // simulation parameters
    public int numPlayers, numCycles,
payoffArray[];

    // ActionGroup for holding an ordered
sequence of action
   public ActionGroup modelActions;


    // the single Schedule
    public Schedule modelSchedule;

    // list of producers
    public List playerList;

    // the world of the economic simulation
    //public Market market;
    public Player playerArray[];
    //public Player player2;
    public Timer timer;
    public RuleMaster ruleMaster;
    public RemoteServer remoteServer;
    public Sum10ObserverSwarm
sum10ObserverSwarm;
```

```
    public int a, b, c, d, e;
    //public int aA1, aA2, aA3, aA4, bA1, bA2,
bA3, bA4, cA1, cA2, cA3, cA4, dA1, dA2, dA3,
dA4;
    //public int aB1, aB2, aB3, aB4, bB1, bB2,
bB3, bB4, cB1, cB2, cB3, cB4, dB1, dB2, dB3,
dB4;
    // data file
    public String dataFile;

    // file for output datas
    public FileWriter file;

    // buffered stream for output datas
    public PrintWriter buff;


    // methods to provide access to the objects
inside the model
    public Player[] getPlayerArray () {
  return playerArray;
    }

    // Constructor of Model Swarm
    public Sum10ModelSwarm (Zone aZone,
Sum10ObserverSwarm o) {
  super (aZone);
  sum10ObserverSwarm = o;

  // simulation parameters
  numPlayers = 27;
  numCycles = 30;

  //payoffs

  a = 0;
  b = 1;
  c = 3;
  d = 4;
  e = 10;

  //data storage file
  dataFile = "data/run1.dat";

  //  customized probe map
  class Sum10ModelProbeMap extends
EmptyProbeMapImpl {
      private VarProbe probeVariable (String
name) {
        return

Globals.env.probeLibrary.getProbeForVariable$in
Class
            (name, Sum10ModelSwarm.this.getClass
());
      }
      private MessageProbe probeMessage (String
name) {
        return
```

```
Globals.env.probeLibrary.getProbeForMessage$inC
lass
            (name, Sum10ModelSwarm.this.getClass
());
      }
      private void addVar (String name) {
        addProbe (probeVariable (name));
      }
      private void addMessage (String name) {
        addProbe (probeMessage (name));
      }
      public Sum10ModelProbeMap (Zone _aZone,
Class aClass) {
        super (_aZone, aClass);
        addVar ("numPlayers");
        addVar ("numCycles");
        addVar ("a");
        addVar ("b");
        addVar ("c");
        addVar ("d");
        addVar ("e");
        addVar ("dataFile");
        //addVar ("payoffArray[1]");
        //addMessage ("addProducer");
        }
      }

  // installing of custom probeMap class
directly into the
  // probeLibrary
  Globals.env.probeLibrary.setProbeMap$For
  (new Sum10ModelProbeMap (aZone, getClass ()),
getClass ());
     }

     // building of model objects
     public Object buildObjects () {
  int i;


  // allow our parent class to build anything
  super.buildObjects();

  payoffArray = new int[5];
  payoffArray[0] = a;
  payoffArray[1] = b;
  payoffArray[2] = c;
  payoffArray[3] = d;
  payoffArray[4] = e;

  // saving datas
  try {

  // file for output datas
  file = new FileWriter (dataFile);

  // build buffered stream for output datas
  buff = new PrintWriter (file);
```

```
  }
  catch (IOException e){
      System.err.println ("Exception in building
data-saving objects: "
                    + e.toString ());
  }
  buff.println ("ID Scelta Punti PuntiTotale
SommaTerna");

  // setting up objects used to represent human
players
  playerList = new ListImpl (getZone ());
  timer = new Timer(numPlayers,
sum10ObserverSwarm, numCycles, buff);
  ruleMaster = new RuleMaster(timer,
numPlayers);
  playerArray = new Player[numPlayers];

  // a loop to create players and put them in
the list
  for (i = 0; i < playerArray.length; i++) {
             playerArray[i] = new Player
(numPlayers, payoffArray, i, playerArray,
timer, ruleMaster);
      System.out.println("Built Player" + i);
      playerList.addLast (playerArray[i]);
  }

  // server class to manage RMI - Remote Method
Invocation
  remoteServer = new RemoteServer(playerArray,
timer, payoffArray);

  return this;
     }

     // Model Schedule
     public Object buildActions () {
  super.buildActions();

  // create the list of simulation actions. We
put these in
  // an action group, because we want these
actions to be
  // executed in a specific order, but these
steps should
  // take no (simulated) timebuild.
  modelActions = new ActionGroupImpl (getZone
());

  // a step to check number of cycles
  try {
      modelActions.createActionTo$message
        (timer, new Selector (timer.getClass (),
"checkCycle", false));
  } catch (Exception e) {
      System.err.println("Exception in
timer.checkCycle: " + e.getMessage ());
  }
```

```
 // main step
 try {
     ActionForEach actionForEach;
     Selector sel =
       new Selector (Class.forName ("Player"),
"step", false);

     actionForEach =
       modelActions.createActionForEach$message
(playerList, sel);
 }
 catch (Exception e) {
     System.err.println("Exception in
player.step: " + e.getMessage ());
 }

 try {
     ActionForEach actionForEach;
     Selector sel =
       new Selector (Class.forName ("Player"),
"step2", false);

     actionForEach =
       modelActions.createActionForEach$message
(playerList, sel);
 }
 catch (Exception e) {
     System.err.println("Exception in
player.step: " + e.getMessage ());
 }

 // a step to make counters for applet-player
synchronization = 0
 try {
     modelActions.createActionTo$message
       (timer, new Selector (timer.getClass (),
"nullBlock", false));
 } catch (Exception e) {
     System.err.println("Exception in
timer.nullBlock: " + e.getMessage ());
 }

 // a step to make the counter of access to
instructions page = 0
 try {
     ActionForEach actionForEach;
     Selector sel =
       new Selector (Class.forName ("Player"),
"nullInstructions", false);

     actionForEach =
       modelActions.createActionForEach$message
(playerList, sel);
 }
 catch (Exception e) {
     System.err.println("Exception in
player.step: " + e.getMessage ());
 }
```

```
 //This schedule has a repeat interval of 1, it
will loop every
 // time step.  The action is executed at time
0 relative to
 // the beginning of the loop.
 // This is a simple schedule, with only three
action that are
 // just repeated every time.
     modelSchedule = new ScheduleImpl (getZone
(), 1);
    modelSchedule.at$createAction (0,
modelActions);

     return this;
    }

    // this model is run as a subswarm of an
observer swarm
    public Activity activateIn (Swarm
swarmContext) {

 // activation of ourselves via the
 // superclass activateIn: method
 super.activateIn (swarmContext);

 // activation of our own schedule
 modelSchedule.activateIn (this);

 // return our activity
 return getActivity ();
    }
}
```

Sum10ObserverSwarm.java

```
import swarm.Globals;
import swarm.Selector;
import swarm.defobj.Zone;
import swarm.defobj.SymbolImpl;

import swarm.activity.Activity;
import swarm.activity.ActionGroup;
import swarm.activity.ActionGroupImpl;
import swarm.activity.Schedule;
import swarm.activity.ScheduleImpl;
import swarm.activity.ActionForEach;

import swarm.collections.List;
import swarm.collections.ListImpl;
```

```
import swarm.objectbase.Swarm;
import swarm.objectbase.SwarmImpl;
import swarm.objectbase.VarProbe;
import swarm.objectbase.MessageProbe;
import swarm.objectbase.EmptyProbeMapImpl;

import java.io.*;

/** The Sum10ModelSwarm encapsulates all the
objects used in the
  simulated world itself (but not the user
interface objects) */

public class Sum10ModelSwarm extends SwarmImpl
{

    // simulation parameters
    public int numPlayers, numCycles,
payoffArray[];

    // ActionGroup for holding an ordered
sequence of action
   public ActionGroup modelActions;


    // the single Schedule
    public Schedule modelSchedule;

    // list of producers
    public List playerList;

    // the world of the economic simulation
    public Player playerArray[];
    public Timer timer;
    public RuleMaster ruleMaster;
    public RemoteServer remoteServer;
    public Sum10ObserverSwarm
sum10ObserverSwarm;
    public int a, b, c, d, e;
    // data file
    public String dataFile;

    // file for output datas
    public FileWriter file;

    // buffered stream for output datas
    public PrintWriter buff;


    // methods to provide access to the objects
inside the model
    public Player[] getPlayerArray () {
 return playerArray;
    }

    // Constructor of Model Swarm
    public Sum10ModelSwarm (Zone aZone,
Sum10ObserverSwarm o) {
```

```
  super (aZone);
  sum10ObserverSwarm = o;

  // simulation parameters
  numPlayers = 27;
  numCycles = 30;

  //payoffs
  a = 0;
  b = 1;
  c = 3;
  d = 4;
  e = 10;

  //data storage file
  dataFile = "data/run1.dat";

  //  customized probe map
  class Sum10ModelProbeMap extends
EmptyProbeMapImpl {
      private VarProbe probeVariable (String
name) {
        return

Globals.env.probeLibrary.getProbeForVariable$in
Class
            (name, Sum10ModelSwarm.this.getClass
());
      }
      private MessageProbe probeMessage (String
name) {
        return

Globals.env.probeLibrary.getProbeForMessage$inC
lass
            (name, Sum10ModelSwarm.this.getClass
());
      }
      private void addVar (String name) {
        addProbe (probeVariable (name));
      }
      private void addMessage (String name) {
        addProbe (probeMessage (name));
      }
      public Sum10ModelProbeMap (Zone _aZone,
Class aClass) {
        super (_aZone, aClass);
        addVar ("numPlayers");
        addVar ("numCycles");

        addVar ("a");
        addVar ("b");
        addVar ("c");
        addVar ("d");
        addVar ("e");
        addVar ("dataFile");
        //addVar ("payoffArray[1]");
        //addMessage ("addProducer");
        }
```

```
      }

  // installing of custom probeMap class
directly into the
  // probeLibrary
  Globals.env.probeLibrary.setProbeMap$For
  (new Sum10ModelProbeMap (aZone, getClass ()),
getClass ());
     }

     // building of model objects
     public Object buildObjects () {
  int i;


  // allow our parent class to build anything
  super.buildObjects();

  /**payoffArray = new int[4][8];

  payoffArray = new int[5];
  payoffArray[0] = a;
  payoffArray[1] = b;
  payoffArray[2] = c;
  payoffArray[3] = d;
  payoffArray[4] = e;

  // saving datas
  try {

  // file for output datas
  file = new FileWriter (dataFile);

  // build buffered stream for output datas
  buff = new PrintWriter (file);
  }
  catch (IOException e){
      System.err.println ("Exception in building
data-saving objects: "
                   + e.toString ());
  }
  buff.println ("ID Scelta Punti PuntiTotale
SommaTerna");

  // setting up objects used to represent human
players
  playerList = new ListImpl (getZone ());
  timer = new Timer(numPlayers,
sum10ObserverSwarm, numCycles, buff);
  ruleMaster = new RuleMaster(timer,
numPlayers);
  playerArray = new Player[numPlayers];

  // a loop to create players and put them in
the list
  for (i = 0; i < playerArray.length; i++) {
            playerArray[i] = new Player
(numPlayers, payoffArray, i, playerArray,
timer, ruleMaster);
```

```
        System.out.println("Built Player" + i);
        playerList.addLast (playerArray[i]);
    }

    // server class to manage RMI - Remote Method
  Invocation
    remoteServer = new RemoteServer(playerArray,
  timer, payoffArray);

    return this;
       }

       // Model Schedule
       public Object buildActions () {
    super.buildActions();

    // create the list of simulation actions. We
  put these in
    // an action group, because we want these
  actions to be
    // executed in a specific order, but these
  steps should
    // take no (simulated) timebuild.
    modelActions = new ActionGroupImpl (getZone
  ());

    // a step to check number of cycles
    try {
        modelActions.createActionTo$message
          (timer, new Selector (timer.getClass (),
  "checkCycle", false));
    } catch (Exception e) {
        System.err.println("Exception in
  timer.checkCycle: " + e.getMessage ());
    }

    // main step
    try {
        ActionForEach actionForEach;
        Selector sel =
          new Selector (Class.forName ("Player"),
  "step", false);

        actionForEach =
          modelActions.createActionForEach$message
  (playerList, sel);
    }
    catch (Exception e) {
        System.err.println("Exception in
  player.step: " + e.getMessage ());
    }

    try {
        ActionForEach actionForEach;
        Selector sel =
          new Selector (Class.forName ("Player"),
  "step2", false);

        actionForEach =
```

```
        modelActions.createActionForEach$message
(playerList, sel);
  }
  catch (Exception e) {
      System.err.println("Exception in
player.step: " + e.getMessage ());
  }

  // a step to make counters for applet-player
synchronization = 0
  try {
      modelActions.createActionTo$message
        (timer, new Selector (timer.getClass (),
"nullBlock", false));
  } catch (Exception e) {
      System.err.println("Exception in timer.nullBlock: " +
e.getMessage ());
  }

  // a step to make the counter of access to
instructions page = 0
  try {
      ActionForEach actionForEach;
      Selector sel =
        new Selector (Class.forName ("Player"),
"nullInstructions", false);

      actionForEach =
        modelActions.createActionForEach$message
(playerList, sel);
  }
  catch (Exception e) {
      System.err.println("Exception in
player.step: " + e.getMessage ());
  }

  //This schedule has a repeat interval of 1, it
will loop every
  // time step.  The action is executed at time
0 relative to
  // the beginning of the loop.
  // This is a simple schedule, with only three
action that are
  // just repeated every time.
      modelSchedule = new ScheduleImpl (getZone
(), 1);
     modelSchedule.at$createAction (0,
modelActions);

      return this;
     }

     // this model is run as a subswarm of an
observer swarm
     public Activity activateIn (Swarm
swarmContext) {
```

```
  // activation of ourselves via the
  // superclass activateIn: method
  super.activateIn (swarmContext);

  // activation of our own schedule
  modelSchedule.activateIn (this);

  // return our activity
  return getActivity ();
     }
}
```

Timer.java

```
import swarm.Globals;
import swarm.Selector;
import swarm.defobj.Zone;
import swarm.defobj.SymbolImpl;

import swarm.activity.Activity;
import swarm.activity.ActionGroup;
import swarm.activity.ActionGroupImpl;
import swarm.activity.Schedule;
import swarm.activity.ScheduleImpl;
import swarm.activity.ActionForEach;

import swarm.collections.List;
import swarm.collections.ListImpl;

import swarm.objectbase.Swarm;
import swarm.objectbase.SwarmImpl;
import swarm.objectbase.VarProbe;
import swarm.objectbase.MessageProbe;
import swarm.objectbase.EmptyProbeMapImpl;

import java.io.*;

/** The Sum10ModelSwarm encapsulates all the
objects used in the
  simulated world itself (but not the user
interface objects) */

public class Sum10ModelSwarm extends SwarmImpl
{

    // simulation parameters
    public int numPlayers, numCycles,
payoffArray[];

    // ActionGroup for holding an ordered
sequence of action
   public ActionGroup modelActions;
```

```
    // the single Schedule
    public Schedule modelSchedule;

    // list of producers
    public List playerList;

    // the world of the economic simulation
    public Player playerArray[];
    public Timer timer;
    public RuleMaster ruleMaster;
    public RemoteServer remoteServer;
    public Sum10ObserverSwarm
sum10ObserverSwarm;
    public int a, b, c, d, e;
    // data file
    public String dataFile;

    // file for output datas
    public FileWriter file;

    // buffered stream for output datas
    public PrintWriter buff;


    // methods to provide access to the objects
inside the model
    public Player[] getPlayerArray () {
  return playerArray;
    }

    // Constructor of Model Swarm
  public Sum10ModelSwarm (Zone aZone,
Sum10ObserverSwarm o) {
  super (aZone);
  sum10ObserverSwarm = o;

  // simulation parameters
  numPlayers = 27;
  numCycles = 30;

  //payoffs
  a = 0;
  b = 1;
  c = 3;
  d = 4;
  e = 10;

  //data storage file
  dataFile = "data/run1.dat";

  //  customized probe map
  class Sum10ModelProbeMap extends
EmptyProbeMapImpl {
      private VarProbe probeVariable (String
name) {
        return
```

```
Globals.env.probeLibrary.getProbeForVariable$in
Class
            (name, Sum10ModelSwarm.this.getClass
());
      }
      private MessageProbe probeMessage (String
name) {
        return
```

Globals.env.probeLibrary.getProbeForMessage$inClass

```
            (name, Sum10ModelSwarm.this.getClass
());
      }
      private void addVar (String name) {
        addProbe (probeVariable (name));
      }
      private void addMessage (String name) {
        addProbe (probeMessage (name));
      }
      public Sum10ModelProbeMap (Zone _aZone,
Class aClass) {
        super (_aZone, aClass);
        addVar ("numPlayers");
        addVar ("numCycles");

        addVar ("a");
        addVar ("b");
        addVar ("c");
        addVar ("d");
        addVar ("e");
        addVar ("dataFile");
        //addVar ("payoffArray[1]");
        //addMessage ("addProducer");
        }
      }

  // installing of custom probeMap class
directly into the
  // probeLibrary
  Globals.env.probeLibrary.setProbeMap$For
  (new Sum10ModelProbeMap (aZone, getClass ()),
getClass ());
     }

     // building of model objects
     public Object buildObjects () {
  int i;


  // allow our parent class to build anything
  super.buildObjects();


  payoffArray = new int[5];
  payoffArray[0] = a;
  payoffArray[1] = b;
```

```
 payoffArray[2] = c;
 payoffArray[3] = d;
 payoffArray[4] = e;

 // saving datas
 try {

 // file for output datas
 file = new FileWriter (dataFile);

 // build buffered stream for output datas
 buff = new PrintWriter (file);
 }
 catch (IOException e){
     System.err.println ("Exception in building
data-saving objects: "
                   + e.toString ());
 }
 buff.println ("ID Scelta Punti PuntiTotale
SommaTerna");

 // setting up objects used to represent human
players
 playerList = new ListImpl (getZone ());
 timer = new Timer(numPlayers,
sum10ObserverSwarm, numCycles, buff);
 ruleMaster = new RuleMaster(timer,
numPlayers);
 playerArray = new Player[numPlayers];

 // a loop to create players and put them in
the list
 for (i = 0; i < playerArray.length; i++) {
            playerArray[i] = new Player
(numPlayers, payoffArray, i, playerArray,
timer, ruleMaster);
     System.out.println("Built Player" + i);
     playerList.addLast (playerArray[i]);
 }

 // server class to manage RMI - Remote Method
Invocation
 remoteServer = new RemoteServer(playerArray,
timer, payoffArray);

 return this;
    }

    // Model Schedule
    public Object buildActions () {
 super.buildActions();

 // create the list of simulation actions. We
put these in
 // an action group, because we want these
actions to be
 // executed in a specific order, but these
steps should
 // take no (simulated) timebuild.
```

```
 modelActions = new ActionGroupImpl (getZone
());

 // a step to check number of cycles
 try {
     modelActions.createActionTo$message
       (timer, new Selector (timer.getClass (),
"checkCycle", false));
 } catch (Exception e) {
     System.err.println("Exception in
timer.checkCycle: " + e.getMessage ());
 }

 // main step
 try {
     ActionForEach actionForEach;
     Selector sel =
       new Selector (Class.forName ("Player"),
"step", false);

     actionForEach =
       modelActions.createActionForEach$message
(playerList, sel);
 }
 catch (Exception e) {
     System.err.println("Exception           in
player.step: " + e.getMessage ());
 }

 try {
     ActionForEach actionForEach;
     Selector sel =
       new Selector (Class.forName ("Player"),
"step2", false);

     actionForEach =
       modelActions.createActionForEach$message
(playerList, sel);
 }
 catch (Exception e) {
     System.err.println("Exception in
player.step: " + e.getMessage ());
 }

 // a step to make counters for applet-player
synchronization = 0
 try {
     modelActions.createActionTo$message
       (timer, new Selector (timer.getClass (),
"nullBlock", false));
 } catch (Exception e) {
     System.err.println("Exception in
timer.nullBlock: " + e.getMessage ());
 }

 // a step to make the counter of access to
instructions page = 0
 try {
```

```
      ActionForEach actionForEach;
      Selector sel =
        new Selector (Class.forName ("Player"),
"nullInstructions", false);

      actionForEach =
        modelActions.createActionForEach$message
(playerList, sel);
  }
  catch (Exception e) {
      System.err.println("Exception in
player.step: " + e.getMessage ());
  }

  //This schedule has a repeat interval of 1, it
will loop every
  // time step.  The action is executed at time
0 relative to
  // the beginning of the loop.
  // This is a simple schedule, with only three
action that are
  // just repeated every time.
      modelSchedule = new ScheduleImpl (getZone
(), 1);
     modelSchedule.at$createAction (0,
modelActions);

      return this;
     }

     // this model is run as a subswarm of an
observer swarm
     public Activity activateIn (Swarm
swarmContext) {

  // activation of ourselves via the
  // superclass activateIn: method
  super.activateIn (swarmContext);

  // activation of our own schedule
  modelSchedule.activateIn (this);

  // return our activity
  return getActivity ();
     }
}
```

Esperimento del 19 aprile 2002[25]

[25] Si tratta della versione integrale dell'esperimento, quella in origine studiata per funzionare su un solo server con 36 umani contemporaneamente.

GT2Instuction.java

```
import java.awt.*;
import java.awt.event.*;

class Gt2Instructions extends Dialog {
    String instructions;
    TextArea l;
    Button button;

    Gt2Instructions(Frame f, String title,
boolean boo, int[][] payoffArray) {
  super(f,title,boo);
  Frame frame = f;
  setLayout(null);
  setBackground(Color.white);
  instructions = ("Instructions to play the
game\n\nIn the first section of the window you
have to make your choice\nand to put your
expectation on the choice of other
player.\nAfter you pushed the Submit button you
have to wait for other\nplayers choices and to
upgrade informations on last round.\nWhen you
obtain your payoff and other data, will run
another\nround and so on.\nRemember that you
can acces this window pushing the last
button\nof the window and that the payoff
matrix is:\n         1     2     3     4\n  1
"+payoffArray[0][0]+","+payoffArray[0][1]+"
"+payoffArray[0][2]+","+payoffArray[0][3]+"
"+payoffArray[0][4]+","+payoffArray[0][5]+"
"+payoffArray[0][6]+","+payoffArray[0][7]+"\n
2  "+payoffArray[1][0]+","+payoffArray[1][1]+"
"+payoffArray[1][2]+","+payoffArray[1][3]+"
"+payoffArray[1][4]+","+payoffArray[1][5]+"
"+payoffArray[1][6]+","+payoffArray[1][7]+"\n
3  "+payoffArray[2][0]+","+payoffArray[2][1]+"
"+payoffArray[2][2]+","+payoffArray[2][3]+"
"+payoffArray[2][4]+","+payoffArray[2][5]+"
"+payoffArray[2][6]+","+payoffArray[2][7]+"\n
4  "+payoffArray[3][0]+","+payoffArray[3][1]+"
"+payoffArray[3][2]+","+payoffArray[3][3]+"
"+payoffArray[3][4]+","+payoffArray[3][5]+"
"+payoffArray[3][6]+","+payoffArray[3][7]);
  l = new TextArea(instructions);
  l.setFont(new Font("Helvetica", Font.PLAIN,
12));
  l.setEditable(false);
  add(l);
  l.setBounds(2,20,396,308);
  button = new Button("Chiudi");
  button.setFont(new
Font("Dialog",Font.BOLD,14));
  add("South", button);
  button.setBounds(175,332,50,24);
  button.addActionListener(new
ButtonHandler2());
```

```
    }

    class ButtonHandler2 implements
ActionListener {
 public void actionPerformed(ActionEvent ev) {
     String s=ev.getActionCommand();
     if ("Chiudi".equals(s)) {
       hide();
     }
 }
    }
}
```

Player.java

```
import swarm.Globals;
import java.io.*;

// The virtual representation of the player

public class Player {

    // paramethers
    public int points, choice, choice2,
payoffArray[], id, cumPoints, accInstructions;
    public int id1, id2, sum3, id3, id4;
    public Player playerArray[];
    public Timer timer;
    public PrintWriter buff;
    public String data;
    public RuleMaster ruleMaster;
    public int numPlayers, changed;

    // constructor
    public Player (int b, int a[], int c,
Player[] pl, Timer t, RuleMaster r) {
 choice = 0;
 points = 0;
 cumPoints = 0;
 accInstructions = 0;
 numPlayers = b;
 payoffArray = a;
 id = c;
 playerArray = pl;
 timer = t;
 ruleMaster = r;
 buff = timer.getBuffer();
 changed = 0;
 int j;
 //choice2 = 1;
      if (id == 0 || id == 3 || id == 6 || id
== 9 || id == 12 || id == 15 || id == 18 || id
== 21 || id == 24 || id == 27 || id == 30 || id
== 33 || id == 36 || id == 39 || id == 42 || id
```

```
== 45 || id == 48 || id == 51 || id == 54 || id
== 57 || id == 60 || id == 63 || id == 66 || id
== 69 || id == 72 || id == 75 || id == 78 || id
== 81 || id == 84 || id == 87 || id == 90 || id
== 93 || id == 96 || id == 99 || id == 102 ||
id == 105 || id == 108 || id == 111 || id ==
114 || id == 117 || id == 120 || id == 123) {
      id1 = id + 1;
      id2 = id + 2;
  }
        if (id == 1 || id == 4 || id == 7 || id
== 10 || id == 13 || id == 16 || id == 19 || id
== 22 || id == 25 || id == 28 || id == 31 || id
== 34 || id == 37 || id == 40 || id == 43 || id
== 46 || id == 49 || id == 52 || id == 55 || id
== 58 || id == 61 || id == 64 || id == 67 || id
== 70 || id == 73 || id == 76 || id == 79 || id
== 82 || id == 85 || id == 88 || id == 91 || id
== 94 || id == 97 || id == 100 || id == 103 ||
id == 106 || id == 109 || id == 112 || id ==
115 || id == 118 || id == 121 || id == 124) {
      id1 = id - 1;
      id2 = id + 1;
  }
  if (id == 2 || id == 5 || id == 8 || id == 11
|| id == 14 || id == 17 || id == 20 || id == 23
|| id == 26 || id == 29 || id == 32 || id == 35
|| id == 38 || id == 41 || id == 44 || id == 47
|| id == 50 || id == 53 || id == 56 || id == 59
|| id == 62 || id == 65 || id == 68 || id == 71
|| id == 74 || id == 77 || id == 80 || id == 83
|| id == 86 || id == 89 || id == 92 || id == 95
|| id == 98 || id == 101 || id == 104 || id ==
107 || id == 110 || id == 113 || id == 116 ||
id == 119 || id == 122 || id == 125) {
      id1 = id - 2;
      id2 = id - 1;
  }
  if ((id >= 0 && id <=2) || (id >= 9 && id
<=11) || (id >= 18 && id <=20) || (id >= 27 &&
id <=29) || (id >= 36 && id <=38) || (id >= 45
&& id <=47) || (id >= 54 && id <=56) || (id >=
63 && id <=65) || (id >= 72 && id <=74) || (id
>= 81 && id <=83) || (id >= 90 && id <=92) ||
(id >= 99 && id <=101) || (id >= 108 && id
<=110) || (id >= 117 && id <=119)) {
      id3 = id + 3;
      id4 = id + 6;
  }
  if ((id >= 3 && id <=5) || (id >= 12 && id
<=14) || (id >= 21 && id <=23) || (id >= 30 &&
id <=32) || (id >= 39 && id <=41) || (id >= 48
&& id <=50) || (id >= 57 && id <=59) || (id >=
66 && id <=68) || (id >= 75 && id <=77) || (id
>= 84 && id <=86) || (id >= 93 && id <=95) ||
(id >= 102 && id <=104) || (id >= 111 && id
<=113) || (id >= 120 && id <=122)) {
      id3 = id - 3;
      id4 = id + 3;
```

```
  }
  if ((id >= 6 && id <=8) || (id >= 15 && id
<=17) || (id >= 24 && id <=26) || (id >= 33 &&
id <=35) || (id >= 42 && id <=44) || (id >= 51
&& id <=53) || (id >= 60 && id <=62) || (id >=
69 && id <=71) || (id >= 78 && id <=80) || (id
>= 87 && id <=89) || (id >= 96 && id <=98) ||
(id >= 105 && id <=107) || (id >= 114 && id
<=116) || (id >= 123 && id <=125)) {
      id3 = id - 6;
      id4 = id - 3;
  }
     }

     // methods
     public int getPoints () {
  return points;
     }

     public int[] getOtherChoice () {
  int i[];
  i = new int[2];
  i[0] = playerArray[id1].getChoice();
  i[1] = playerArray[id2].getChoice();
  return i;
     }

     public Object setChoice (int i) {
  choice = i;
  timer.setBlock(id);
  return this;
     }

     public int getChoice () {
  return choice;
     }

     public int getChoice2 () {
  return choice2;
     }

     public Object setChoice2 () {
  choice2 = choice;
  return this;
     }

     public int getCumPoints () {
  return cumPoints;
     }

     public Object setInstructions () {
  accInstructions++;
  return this;
     }

     public Object nullInstructions () {
  accInstructions = 0;
  return this;
     }
```

```
    public int getSum3 () {
  int sum3b, i[];
  i = getOtherChoice();
  sum3b = choice + i[0] + i[1];
  return sum3b;
    }

    // Schedule's method
    public Object step () {
      if (id == 0 || id == 3 || id == 6 || id
== 9 || id == 12 || id == 15 || id == 18 || id
== 21 || id == 24 || id == 27 || id == 30 || id
== 33 || id == 36 || id == 39 || id == 42 || id
== 45 || id == 48 || id == 51 || id == 54 || id
== 57 || id == 60 || id == 63 || id == 66 || id
== 69 || id == 72 || id == 75 || id == 78 || id
== 81 || id == 84 || id == 87 || id == 90 || id
== 93 || id == 96 || id == 99 || id == 102 ||
id == 105 || id == 108 || id == 111 || id ==
114 || id == 117 || id == 120 || id == 123) {
  }
      else {
  setChoice(ruleMaster.getChoice(this));
      }
  return this;
    }

    public Object step2 () {
  while (!(timer.askBlock())) {
      continue;
  }


  //Upgrade payoffs
  //int otherChoice[];
  //otherChoice = getOtherChoice ();

  /*sum3 = getSum3();
  if (sum3 == 10)
      points = 40 - choice;
  if (sum3 > 10)
      points = 30 - choice;
  if (sum3 < 10)
      points = -choice;*/

  //if (otherChoice == expectation)
  //  payoff ++;
  /*I SISTEMI DI REMUNERAZIONE
  - gruppo A1 (metà del gruppo A) e B1 (metà del
gruppo B): vincite definite
  individualmente e pari a:

  se somma =10; 40- numero dichiarato
  se somma>10; 20-numero di dichiarato (l'anno
scorso era 30-numero
  dichiarato, ma ora penalizzerei maggiormente
questo risultato)
  se somma<10; 0- numero dichiarato
```

```
 - gruppo A2 e B2: vincite uguali per tutto il
gruppo e pari a:

 come per A1 e B!, ma invece del numero
dichiarato da ciascuno si sottrae la
 media dei numeri dichiarati dai tre (se
necessario approssimata all'intero
 più vicino, altrimenti approssimata ad un
decimale)*/

 if((this.id<27) || (this.id>=54 &&
this.id<81))
 {
       sum3 = getSum3();
       if (sum3 == 10)
             points = 40 - choice;
       if (sum3 > 10)
             points = 20 - choice;
       if (sum3 < 10)
       points = - choice;
 }
 if((this.id>=27 && this.id<54) ||
(this.id>=81))
 {
       sum3 = getSum3();
       if (sum3 == 10)
             points = 40 - Math.round(sum3/3);
       if (sum3 > 10)
             points = 20 - Math.round(sum3/3);
       if (sum3 < 10)
             points = - Math.round(sum3/3);
 }
 if (changed == 1) {
     changed = 0;
     cumPoints = 0;
 }
 if (timer.askChange()) {
    changed = 1;
    id1 = id3;
    id2 = id4;
 }
 cumPoints += points;
 timer.setBlock2();
 setChoice2();
 data = (String) (id + " " + choice + " " +
points + " " + cumPoints + " " + sum3);
 //System.out.println(data);
 buff.println(data);
 return this;
    }
}
```

RemoteServer.java

```
import java.rmi.Naming;
import java.rmi.*;
import java.net.*;
import java.io.*;
import java.rmi.server.*;
import java.rmi.registry.LocateRegistry;
// Remote Server's class easily manages in
Swarm simulation
// RMI Java technology

public class RemoteServer {
    Player[] playerArray;
    Timer timer;
    private static RMIRegistry  rmi;
    public RemoteServer(Player[] pl, Timer t,
int[] payoffArray) {
 playerArray = pl;
 timer = t;
 try {
     activateRegistry();
     //System.setSecurityManager(new
MySecurityManager());
     RemoteSum10 c = new
RemoteSum10Impl(playerArray, timer,
payoffArray);
     // Pay attention to the IP of your host
and to the port:
     // default port of RMIRegistry is 1099

Naming.rebind("//130.192.198.42:1098/RemoteSum1
0", c);
     System.out.println("RemoteServer is
registered.");
 } catch (Exception e) {
     System.out.println("Trouble in get
RemoteSum10: " + e);
 }
    }
    public void activateRegistry() {


 try
     {
       rmi = new RMIRegistry();
     }
 catch ( java.rmi.UnknownHostException uhe )
     {
       System.out.println( "The host computer
name you have specified, does not match your
real computer name." );
     }
 catch ( RemoteException re )
     {
       System.out.println( "Error starting
service" );
       System.out.println( "" + re );
     }
 catch ( MalformedURLException mURLe )
```

```
      {
        System.out.println( "Internal error" +
mURLe );
      }
  catch ( NotBoundException nbe )
      {
        System.out.println( "Not Bound" );
        System.out.println( "" + nbe );
      }
    }
}
```

RemoteSum10.java

```
import java.rmi.*;

// Interface to Server Methods

public interface RemoteSum10 extends Remote {
    public void setChoice(int i, int v) throws
RemoteException;
    public int getCumPoints(int i) throws
RemoteException;
    public boolean askRound(int i) throws
RemoteException;
    public boolean askBlock2() throws
RemoteException;
    public int getPoints(int i) throws
RemoteException;
    public int[] getPayoffArray() throws
RemoteException;
    public int getChoice(int i) throws
RemoteException;
    public boolean askEnd() throws
RemoteException;
    public int askNumRound() throws
RemoteException;
    public void setBlock3() throws
RemoteException;
    public boolean askChange() throws
RemoteException;
}
```

RemoteSum10Impl.java

```
import java.rmi.*;
import java.rmi.server.*;
import java.util.*;
```

```
// Implementation of Server's Methods.

public class RemoteSum10Impl extends
UnicastRemoteObject
    implements RemoteSum10 {
    public Player[] playerArray;
    public Timer timer;
    public int[] payoffArray;
    public RemoteSum10Impl(Player[] pl, Timer
t, int[] p) throws RemoteException {
 super();
 playerArray = pl;
 timer = t;
 payoffArray = p;
    }
    public void setChoice(int i, int v) throws
RemoteException {
 int id = i, value = v;
 playerArray[id].setChoice(value);
    }
    public int getCumPoints(int i) throws
RemoteException  {
 int id = i;
 int value = playerArray[id].getCumPoints();
 return value;
 }
    public boolean askRound(int i) throws
RemoteException  {
 int id = i;
 boolean ok = timer.getRound(id);
 return ok;
    }
    public void setBlock3() throws
RemoteException  {
 timer.setBlock3();
    }
    public boolean askBlock2() throws
RemoteException  {
 boolean ok = (timer.askBlock2());
 return ok;
 }
    public int getPoints(int i) throws
RemoteException  {
 int id = i;
 int value = playerArray[id].getPoints();
 return value;
    }
    public int[] getPayoffArray() throws
RemoteException  {
 return payoffArray;
    }

    public int getChoice(int i) throws
RemoteException  {
 int value = playerArray[i].getChoice2();
 return value;
    }
```

```
    public boolean askEnd() throws
RemoteException  {
 boolean ok;
 if (timer.cycles >= (timer.numCycles - 1))
     ok = true;
 else
     ok = false;
 return ok;
    }
    public boolean askChange() throws
RemoteException  {
 boolean ok;
 if (timer.askChange())
     ok = true;
 else
     ok = false;
 return ok;
    }
    public int askNumRound() throws
RemoteException  {
 int i = timer.cycles;
 return i;
    }
}
```

RMIRegistry.java

```
import java.rmi.*;
import java.rmi.server.*;
import java.util.*;

// Implementation of Server's Methods.

public class RemoteSum10Impl extends
UnicastRemoteObject
    implements RemoteSum10 {
    public Player[] playerArray;
    public Timer timer;
    public int[] payoffArray;
    public RemoteSum10Impl(Player[] pl, Timer
t, int[] p) throws RemoteException {
 super();
 playerArray = pl;
 timer = t;
 payoffArray = p;
    }
    public void setChoice(int i, int v) throws
RemoteException {
 int id = i, value = v;
 playerArray[id].setChoice(value);
    }
    public int getCumPoints(int i) throws
RemoteException  {
 int id = i;
```

```
  int value = playerArray[id].getCumPoints();
  return value;
  }
     public boolean askRound(int i) throws
RemoteException  {
  int id = i;
  boolean ok = timer.getRound(id);
  return ok;
     }
     public void setBlock3() throws
RemoteException  {
  timer.setBlock3();
     }
     public boolean askBlock2() throws
RemoteException  {
  boolean ok = (timer.askBlock2());
  return ok;
  }
     public int getPoints(int i) throws
RemoteException  {
  int id = i;
  int value = playerArray[id].getPoints();
  return value;
     }
     public int[] getPayoffArray() throws
RemoteException  {
  return payoffArray;
     }

     public int getChoice(int i) throws
RemoteException  {
  int value = playerArray[i].getChoice2();
  return value;
     }
     public boolean askEnd() throws
RemoteException  {
  boolean ok;
  if (timer.cycles >= (timer.numCycles - 1))
      ok = true;
  else
      ok = false;
  return ok;
     }
     public boolean askChange() throws
RemoteException  {
  boolean ok;
  if (timer.askChange())
      ok = true;
  else
      ok = false;
  return ok;
     }
     public int askNumRound() throws
RemoteException  {
  int i = timer.cycles;
  return i;
     }
}
```

RuleMaster.java

```
import swarm.Globals;
//import java.io.*;

// Rule Master

public class RuleMaster {

    // paramethers
    public Timer timer;
    public int numPlayers;
    // constructor
    public RuleMaster (Timer t, int r) {
      timer = t;
      numPlayers = r;
  //int i, j, z, x;
    }

    // methods
    public int getChoice (Player a)
    {
      int b = 0;
      int j;



// agenti altruisti

      /*if (a.id == 4 || a.id == 7 || a.id ==
13 || a.id == 19 || a.id == 25 || a.id == 31 ||
a.id == 37 || a.id == 43 || a.id == 49 || a.id
== 55 || a.id == 61 || a.id == 67 || a.id == 73
|| a.id == 79 || a.id == 85 || a.id == 91 ||
a.id == 97 || a.id == 103 || a.id == 109 ||
a.id == 115 || a.id == 121) {
            b = 4;
      }

      if (a.id == 5 || a.id == 8 || a.id == 14
|| a.id == 20 || a.id == 26 || a.id == 32 ||
a.id == 38 || a.id == 38 || a.id == 44 || a.id
== 50 || a.id == 56 || a.id == 62 || a.id == 68
|| a.id == 74 || a.id == 80 || a.id == 86 ||
a.id == 92 || a.id == 98 || a.id == 104 || a.id
== 110 || a.id == 116 || a.id == 122) {
        if (timer.cycles == 0) {
          b = 3;
        }
        else {
            if ((a.playerArray[a.id1].choice2 +
a.playerArray[a.id2].choice2) < 10) {
```

```
                if
((a.playerArray[a.id1].choice2 +
a.playerArray[a.id2].choice2 + 1) < 10){
                  if
((a.playerArray[a.id1].choice2 +
a.playerArray[a.id2].choice2 + 3) < 10){
                    if
((a.playerArray[a.id1].choice2 +
a.playerArray[a.id2].choice2 + 4) < 10){
                      b = 10;
                    }
                    else
                      b = 4;
                  }
                  else
                    b = 3;
                }
                else
                  b = 1;
            }
            else
               b = 0;

        }
      }

// agenti egoisti

      if (a.id == 1 || a.id == 10 || a.id == 16
|| a.id == 22 || a.id == 28 || a.id == 34 ||
a.id == 40 || a.id == 46 || a.id == 52 || a.id
== 58 || a.id == 64 || a.id == 70 || a.id == 76
|| a.id == 82 || a.id == 88 || a.id == 94 ||
a.id == 100 || a.id == 106 || a.id == 112 ||
a.id == 118 || a.id == 124) {
            b = 3;
      }

      if (a.id == 2 || a.id == 11 || a.id == 17
|| a.id == 23 || a.id == 29 || a.id == 35 ||
a.id == 41 || a.id == 47 || a.id == 53 || a.id
== 59 || a.id == 65 || a.id == 71 || a.id == 77
|| a.id == 83 || a.id == 89 || a.id == 95 ||
a.id == 101 || a.id == 107 || a.id == 113 ||
a.id == 119 || a.id == 125) {
        if (timer.cycles == 0) {
          b = 3;
        }
        else {
            if (((a.playerArray[a.id1].choice2
+ a.playerArray[a.id2].choice2) < 7 ||
(a.playerArray[a.id1].choice2 +
a.playerArray[a.id2].choice2) > 10)){
                  b = 0;
            }
            else
               b = 3;
        }
      }
```

```
 return b;*/
 // divisione per 3 (%da' il resto della
divisione)

       if((a.id%3==1) && (a.id<54))
       {
             //egoista debole

             if (timer.cycles == 0)
             {
                   b = 3;
             }
             else
             {
                   if
(((a.playerArray[a.id1].choice2 +
a.playerArray[a.id2].choice2) < 7 ||
(a.playerArray[a.id1].choice2 +
a.playerArray[a.id2].choice2) > 10))
                   {
                         b = 0;
                   }
                   else
                         b = 3;
             }
       }

       if((a.id%3==2) && (a.id<54))
       {
             //egoista forte

             b = 3;
       }

       if((a.id%3==1) && (a.id>54))
       {
             //altruista debole
             if (timer.cycles == 0)
             {
                b = 3;
             }
             else
             {
                   if
((a.playerArray[a.id1].choice2 +
a.playerArray[a.id2].choice2) < 10)
                   {
                         if
((a.playerArray[a.id1].choice2 +
a.playerArray[a.id2].choice2 + 1) < 10)
                         {
                               if
((a.playerArray[a.id1].choice2 +
a.playerArray[a.id2].choice2 + 3) < 10)
                               {
                                     if
((a.playerArray[a.id1].choice2 +
a.playerArray[a.id2].choice2 + 4) < 10)
                                     {
```

```
                                                b =
10;
                                        }
                                        else
                                                b =
4;
                                }
                                else
                                b = 3;
                        }
                        else
                                b = 1;
                }
                else
                   b = 0;

                }
        }

        if((a.id%3==2) && (a.id>54))
        {
                //altruista forte

                b=4;
        }
        return b;
    }
}
```

StartSum10.java

```
// Game Theory - Model 1 Copyright © 2000
Riccardo Boero
// This library is distributed without any
warranty; without even the
// implied warranty of merchantability or
fitness for a particular purpose.
// See file LICENSE for details and terms of
copying.

import swarm.Globals;

public class StartSum10 {
    // Main function
    public static void main (String[] args) {
        Globals.env.initSwarm ("sum10",
"2.1.1", "bug-swarm@santafe.edu", args);
```

```
 Sum10ObserverSwarm topLevelSwarm =
     new Sum10ObserverSwarm
(Globals.env.globalZone);
 Globals.env.setWindowGeometryRecordName
(topLevelSwarm, "topLevelSwarm");

 topLevelSwarm.buildObjects ();
 topLevelSwarm.buildActions ();
 topLevelSwarm.activateIn (null);
 topLevelSwarm.go ();
 topLevelSwarm.drop ();
    }
}
```

Sum10Client.java

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
import java.net.*;
import java.rmi.*;
import java.rmi.Naming;

public class Sum10Client extends Applet  {
    int value, value2, id, id1, id2, id3, id4;
//payoffArray[];
    String identification, valueS;
    boolean updated, activated;
    //Object parentFrame;
    RemoteSum10 server;

    java.awt.Label label1 = new
java.awt.Label();

    /*java.awt.Checkbox radioButton1 = new
java.awt.Checkbox();
    java.awt.CheckboxGroup Group1 = new
java.awt.CheckboxGroup();
    java.awt.Checkbox radioButton2 = new
java.awt.Checkbox();
    java.awt.Checkbox radioButton3 = new
java.awt.Checkbox();
    java.awt.Checkbox radioButton4 = new
java.awt.Checkbox();
    java.awt.Checkbox radioButton5 = new
java.awt.Checkbox();

    java.awt.Label label2 = new
java.awt.Label();
    java.awt.Label label11 = new
java.awt.Label();
    java.awt.Label label12 = new
java.awt.Label();
```

```
    java.awt.Label label13 = new
java.awt.Label();
    java.awt.Label label14 = new
java.awt.Label();

    java.awt.TextField textField7 = new
java.awt.TextField();
    java.awt.TextField textField8 = new
java.awt.TextField();
    java.awt.TextField textField9 = new
java.awt.TextField();
    java.awt.TextField textField10 = new
java.awt.TextField();
    java.awt.TextField textField11 = new
java.awt.TextField();*/

    Choice CBScelta=new Choice();

    java.awt.TextField textField12 = new
java.awt.TextField();
    java.awt.TextField textField13 = new
java.awt.TextField();
    java.awt.Label label3 = new
java.awt.Label();
    java.awt.Button button1 = new
java.awt.Button();
    java.awt.Label label4 = new
java.awt.Label();
    java.awt.TextField textField1 = new
java.awt.TextField();
    java.awt.Label label5 = new
java.awt.Label();
    java.awt.TextField textField2 = new
java.awt.TextField();
    java.awt.Label label6 = new
java.awt.Label();
    java.awt.TextField textField3 = new
java.awt.TextField();
    java.awt.Label label7 = new
java.awt.Label();
    java.awt.TextField textField4 = new
java.awt.TextField();
    java.awt.Label label8 = new
java.awt.Label();
    java.awt.Label label9 = new
java.awt.Label();
    java.awt.TextField textField5 = new
java.awt.TextField();
    java.awt.Label label15 = new
java.awt.Label();
    java.awt.Label label16 = new
java.awt.Label();
    java.awt.Label label10 = new
java.awt.Label();
    java.awt.TextField textField6 = new
java.awt.TextField();
    java.awt.TextArea textArea1 = new
java.awt.TextArea();
```

```
    java.awt.Button button2 = new
java.awt.Button();
    //aggiunte
    java.awt.Label label17 = new
java.awt.Label();
    java.awt.TextField textField14 = new
java.awt.TextField();

    public void init() {
  identification =
getParameter("identification");
  id = Integer.parseInt(identification);
  setLayout(null);
  setSize(364,398);
  activated = false;
  label1.setText("Scelta:");
  add(label1);
  label1.setFont(new Font("Dialog", Font.BOLD,
12));
  label1.setBounds(12,12,50,12);

  add(CBScelta);
  CBScelta.setFont(new Font("Dialog", Font.BOLD,
12));
  CBScelta.setBounds(70,12,50,12);
  CBScelta.add("0");
  CBScelta.add("1");
  CBScelta.add("2");
  CBScelta.add("3");
  CBScelta.add("4");
  CBScelta.add("5");
  CBScelta.add("6");
  CBScelta.add("7");
  CBScelta.add("8");
  CBScelta.add("9");
  CBScelta.add("10");

  /*radioButton1.setCheckboxGroup(Group1);
  radioButton1.setLabel("A");
  add(radioButton1);
  radioButton1.setFont(new Font("Dialog",
Font.BOLD, 12));
  radioButton1.setBounds(70,12,28,12);
  radioButton2.setCheckboxGroup(Group1);
  radioButton2.setLabel("B");
  add(radioButton2);
  radioButton2.setFont(new Font("Dialog",
Font.BOLD, 12));
  radioButton2.setBounds(126,12,28,12);
  radioButton3.setCheckboxGroup(Group1);
  radioButton3.setLabel("C");
  add(radioButton3);
  radioButton3.setFont(new Font("Dialog",
Font.BOLD, 12));
  radioButton3.setBounds(182,12,28,12);
  radioButton4.setCheckboxGroup(Group1);
  radioButton4.setLabel("D");
  add(radioButton4);
```

```
  radioButton4.setFont(new Font("Dialog",
Font.BOLD, 12));
  radioButton4.setBounds(238,12,28,12);
  radioButton5.setCheckboxGroup(Group1);
  radioButton5.setLabel("E");
  add(radioButton5);
  radioButton5.setFont(new Font("Dialog",
Font.BOLD, 12));
  radioButton5.setBounds(294,12,28,12);
  label2.setText("A:");
  label2.setAlignment(java.awt.Label.RIGHT);
  add(label2);
  label2.setBounds(50,36,20,24);
  add(textField7);
  textField7.setEditable(false);
  textField7.setBounds(70,36,30,24);
  label11.setText("B:");
  label11.setAlignment(java.awt.Label.RIGHT);
  add(label11);
  label11.setBounds(106,36,20,24);
  add(textField8);
  textField8.setEditable(false);
  textField8.setBounds(126,36,30,24);
  label12.setText("C:");
  label12.setAlignment(java.awt.Label.RIGHT);
  add(label12);
  label12.setBounds(162,36,20,24);
  add(textField9);
  textField9.setEditable(false);
  textField9.setBounds(182,36,30,24);
  label13.setText("D:");
  label13.setAlignment(java.awt.Label.RIGHT);
  add(label13);
  label13.setBounds(218,36,20,24);
  add(textField10);
  textField10.setEditable(false);
  textField10.setBounds(238,36,30,24);
  label14.setText("E:");
  label14.setAlignment(java.awt.Label.RIGHT);
  add(label14);
  label14.setBounds(274,36,20,24);
  add(textField11);
  textField11.setEditable(false);
  textField11.setBounds(294,36,30,24);*/



  //modificate
  label15.setText("Somma della tua terna:");
  label15.setAlignment(java.awt.Label.RIGHT);
  add(label15);
  label15.setBounds(12,218,130,24);
  add(textField12);
  textField12.setEditable(false);
  textField12.setBounds(150,218,40,24);
  label16.setText("Somma tuoi punti:");
  label16.setAlignment(java.awt.Label.RIGHT);
  add(label16);
  label16.setBounds(200,218,100,24);
```

```
 add(textField13);
 textField13.setEditable(false);
 textField13.setBounds(310,218,40,24);
 //fine
 //aggiunte
 label17.setText("Turno n.");
 label17.setAlignment(java.awt.Label.RIGHT);
 add(label17);
 label17.setBounds(120,184,50,24);
 add(textField14);
 textField14.setEditable(false);
 textField14.setBounds(170,184,40,24);

 label3.setText("Risultati dell\'ultimo
incontro:");
 add(label3);
label3.setFont(new Font("Dialog",
Font.BOLD|Font.ITALIC,12));
 label3.setBounds(12,96,180,24);
 button1.setLabel("Invia i dati");
 add(button1);
 button1.setBackground(java.awt.Color.lightGray
);
 button1.setBounds(12,70,340,24);
 button1.addActionListener(new
ButtonHandler());
 label4.setText("Tua scelta:");
 label4.setAlignment(java.awt.Label.RIGHT);
 add(label4);
 label4.setBounds(12,120,83,24);
 add(textField1);
 textField1.setEditable(false);
 textField1.setBounds(95,120,40,24);
 label5.setText("Scelta di 1:");
 label5.setAlignment(java.awt.Label.RIGHT);
 add(label5);
 label5.setBounds(135,120,65,24);
 add(textField2);
 textField2.setEditable(false);
 textField2.setBounds(200,120,40,24);
 label6.setText("Scelta di 2:");
 label6.setAlignment(java.awt.Label.RIGHT);
 add(label6);
 label6.setBounds(240,120,72,24);
 add(textField3);
 textField3.setEditable(false);
 textField3.setBounds(312,120,40,24);

 label7.setText("Tuo punteggio:");
 label7.setAlignment(java.awt.Label.RIGHT);
 add(label7);
 label7.setBounds(12,156,83,24);
 add(textField4);
 textField4.setEditable(false);
 textField4.setBounds(95,156,40,24);
 label9.setText("Punti di 1:");
 label9.setAlignment(java.awt.Label.RIGHT);
 add(label9);
 label9.setBounds(135,156,65,24);
```

```
  add(textField5);
  textField5.setEditable(false);
  textField5.setBounds(200,156,40,24);
  label10.setText("Punti di 2:");
  label10.setAlignment(java.awt.Label.RIGHT);
  add(label10);
  label10.setBounds(240,156,72,24);
  add(textField6);
  textField6.setEditable(false);
  textField6.setBounds(312,156,40,24);

  button2.setLabel("Attiva");
  add(button2);
  button2.setBackground(java.awt.Color.lightGray
);
  button2.setBounds(12,244,340,24);
  button2.addActionListener(new
ButtonHandler());

  /**button2.setLabel("Fate la vostra
giocata!");
  add(button2);
  button2.setBackground(java.awt.Color.yellow);
  button2.setBounds(12,244,340,24);
  button2.addActionListener(new
ButtonHandler());*/
  label8.setText("Messaggi:");
  add(label8);
label8.setFont(new Font("Dialog",
Font.BOLD|Font.ITALIC, 12));
  label8.setBounds(12,268,132,24);
  add(textArea1);
  textArea1.setBounds(12,292,340,84);

  updated = true;


      if (id == 0 || id == 3 || id == 6 || id ==
9 || id == 12 || id == 15 || id == 18 || id ==
21 || id == 24 || id == 27 || id == 30 || id ==
33 || id == 36 || id == 39 || id == 42 || id ==
45 || id == 48 || id == 51 || id == 54 || id ==
57 || id == 60 || id == 63 || id == 66 || id ==
69 || id == 72 || id == 75 || id == 78 || id ==
81 || id == 84 || id == 87 || id == 90 || id ==
93 || id == 96 || id == 99 || id == 102 || id
== 105 || id == 108 || id == 111 || id == 114
|| id == 117 || id == 120 || id == 123) {
      id1 = id + 1;
      id2 = id + 2;
  }
        if (id == 1 || id == 4 || id == 7 || id
== 10 || id == 13 || id == 16 || id == 19 || id
== 22 || id == 25 || id == 28 || id == 31 || id
== 34 || id == 37 || id == 40 || id == 43 || id
== 46 || id == 49 || id == 52 || id == 55 || id
== 58 || id == 61 || id == 64 || id == 67 || id
== 70 || id == 73 || id == 76 || id == 79 || id
== 82 || id == 85 || id == 88 || id == 91 || id
```

```
== 94 || id == 97 || id == 100 || id == 103 ||
id == 106 || id == 109 || id == 112 || id ==
115 || id == 118 || id == 121 || id == 124) {
      id1 = id - 1;
      id2 = id + 1;
  }
  if (id == 2 || id == 5 || id == 8 || id == 11
|| id == 14 || id == 17 || id == 20 || id == 23
|| id == 26 || id == 29 || id == 32 || id == 35
|| id == 38 || id == 41 || id == 44 || id == 47
|| id == 50 || id == 53 || id == 56 || id == 59
|| id == 62 || id == 65 || id == 68 || id == 71
|| id == 74 || id == 77 || id == 80 || id == 83
|| id == 86 || id == 89 || id == 92 || id == 95
|| id == 98 || id == 101 || id == 104 || id ==
107 || id == 110 || id == 113 || id == 116 ||
id == 119 || id == 122 || id == 125) {
      id1 = id - 2;
      id2 = id - 1;
  }
  if ((id >= 0 && id <=2) || (id >= 9 && id
<=11) || (id >= 18 && id <=20) || (id >= 27 &&
id <=29) || (id >= 36 && id <=38) || (id >= 45
&& id <=47) || (id >= 54 && id <=56) || (id >=
63 && id <=65) || (id >= 72 && id <=74) || (id
>= 81 && id <=83) || (id >= 90 && id <=92) ||
(id >= 99 && id <=101) || (id >= 108 && id
<=110) || (id >= 117 && id <=119)) {
      id3 = id + 3;
      id4 = id + 6;
  }
  if ((id >= 3 && id <=5) || (id >= 12 && id
<=14) || (id >= 21 && id <=23) || (id >= 30 &&
id <=32) || (id >= 39 && id <=41) || (id >= 48
&& id <=50) || (id >= 57 && id <=59) || (id >=
66 && id <=68) || (id >= 75 && id <=77) || (id
>= 84 && id <=86) || (id >= 93 && id <=95) ||
(id >= 102 && id <=104) || (id >= 111 && id
<=113) || (id >= 120 && id <=122)) {
      id3 = id - 3;
      id4 = id + 3;
  }
  if ((id >= 6 && id <=8) || (id >= 15 && id
<=17) || (id >= 24 && id <=26) || (id >= 33 &&
id <=35) || (id >= 42 && id <=44) || (id >= 51
&& id <=53) || (id >= 60 && id <=62) || (id >=
69 && id <=71) || (id >= 78 && id <=80) || (id
>= 87 && id <=89) || (id >= 96 && id <=98) ||
(id >= 105 && id <=107) || (id >= 114 && id
<=116) || (id >= 123 && id <=125)) {
      id3 = id - 6;
      id4 = id - 3;
  }
     }
    public boolean askRound(int i){
  boolean aa = false;
  try  {
      aa = server.askRound(i);
  } catch (Exception ex) {
```

```
      textArea1.setText(ex.toString());
  }
  return aa;
  }
     public boolean askBlock2(){
  boolean aa = true;
  try  {
      aa = server.askBlock2();
  } catch (Exception ex) {
      textArea1.setText(ex.toString());
  }
  return aa;
     }

     public void update () {
  try  {
                    //Update information

textField1.setText(Integer.toString(value));
      int newChoice1 = server.getChoice(id1);

textField2.setText(Integer.toString(newChoice1)
);
      int newChoice2 = server.getChoice(id2);

textField3.setText(Integer.toString(newChoice2)
);
      int newPoints = server.getPoints(id);

textField4.setText(Integer.toString(newPoints))
;
      int newPoints1 = server.getPoints(id1);

textField5.setText(Integer.toString(newPoints1)
);
      int newPoints2 = server.getPoints(id2);

textField6.setText(Integer.toString(newPoints2)
);

textField12.setText(Integer.toString(value+newC
hoice1+newChoice2));
      int newCumPoints =
server.getCumPoints(id);

textField13.setText(Integer.toString(newCumPoin
ts));
                    //aggiunte
      int newNumRound = server.askNumRound();

textField14.setText(Integer.toString(newNumRoun
d + 1));

      updated = true;
      if (server.askEnd())
        textArea1.setText(" -- L'ESPERIMENTO E'
TERMINATO!!! --\n -- L'ESPERIMENTO E'
TERMINATO!!! --\n -- L'ESPERIMENTO E'
TERMINATO!!! --");
```

```
      else {
        if (server.askChange()) {
            textArea1.setText(" -- ATTENZIONE!
Da ora affronti nuovi giocatori! --\n --
ATTENZIONE! Da ora affronti nuovi giocatori! --
\n -- ATTENZIONE! Da ora affronti nuovi
giocatori! --");
            id1 = id3;
            id2 = id4;
        }else
            textArea1.setText("Risultati
aggiornati, ora puoi effettuare una nuova
scelta.");
      }
  } catch (Exception ex) {
      textArea1.setText(ex.toString());
  }
     }

     public void activate () {
  try {
      URL hostURL = getCodeBase();
      String host = hostURL.getHost();
      // Change the port number if necessary
      server =
(RemoteSum10)Naming.lookup("//"+host+":1098/Rem
oteSum10");
      } catch (Exception ex) {
      textArea1.setText(ex.toString());
      }
  /*try  {
      payoffArray = server.getPayoffArray();
  } catch (Exception ex) {
      textArea1.setText(ex.toString());
  }*/

  // A message to tell players that the first
round is an example
  //textArea1.setText("ATTENZIONE:\nil primo
turno e' di prova!!!");
  /*textField7.setText(Integer.toString(payoffAr
ray[0]));
  textField8.setText(Integer.toString(payoffArra
y[1]));
  textField9.setText(Integer.toString(payoffArra
y[2]));
  textField10.setText(Integer.toString(payoffArr
ay[3]));
  textField11.setText(Integer.toString(payoffArr
ay[4]));*/

  activated = true;
     }

     class ButtonHandler implements
ActionListener {
  public void actionPerformed(ActionEvent ev) {
      String s=ev.getActionCommand();
```

```
      if ("Invia i dati".equals(s)) {
        /*if (Group1.getSelectedCheckbox() ==
null) {
            textArea1.setText("Occorre
selezionare la propria scelta.");
        }
        else {*/
            if (!(askRound(id))) {
              if (updated) {
                  try  {
                    //Submit values
                    //valueS =
Group1.getSelectedCheckbox().getLabel();
                    /*if (valueS == "A")
                        value = payoffArray[0];
                    if (valueS == "B")
                        value = payoffArray[1];
                    if (valueS == "C")
                        value = payoffArray[2];
                    if (valueS == "D")
                        value = payoffArray[3];
                    if (valueS == "E")
                        value =
payoffArray[4];*/


  value=CBScelta.getSelectedIndex();
                    server.setChoice(id, value);
                    updated = false;
                    textArea1.setText("Ok,
adesso puoi provare ad aggiornare i
risultati.");
                  } catch (Exception ex) {

  textArea1.setText(ex.toString());
                  }
              }
              else {
                  textArea1.setText("Prima devi
aggiornare i risultati!!!");
              }
            }
            else {
              textArea1.setText("Prima devi
aggiornare i risultati!!!");
            }
        //}
      }


      if ("Attiva".equals(s)) {
        activate();
        textArea1.setText("Ora puoi iniziare a
giocare.");
        button2.setLabel("Aggiorna i
risultati");
      }
      if ("Aggiorna i risultati".equals(s)) {
        if ((askRound(id))) {
```

```
            if ((askBlock2())) {
              if (!(updated)) {
                  try  {
                    //Update information

 textField1.setText(Integer.toString(value));
                    int newChoice1 =
server.getChoice(id1);

 textField2.setText(Integer.toString(newChoice1
));
                    int newChoice2 =
server.getChoice(id2);

 textField3.setText(Integer.toString(newChoice2
));
                    int newPoints =
server.getPoints(id);

 textField4.setText(Integer.toString(newPoints)
);
                    int newPoints1 =
server.getPoints(id1);

 textField5.setText(Integer.toString(newPoints1
));
                    int newPoints2 =
server.getPoints(id2);

 textField6.setText(Integer.toString(newPoints2
));

 textField12.setText(Integer.toString(value+new
Choice1+newChoice2));
                    int newCumPoints =
server.getCumPoints(id);

 textField13.setText(Integer.toString(newCumPoi
nts));
                    //aggiunte
                    int newNumRound =
server.askNumRound() + 1;

 textField14.setText(Integer.toString(newNumRou
nd));

                    updated = true;
                    if (server.askEnd())
                        textArea1.setText(" --
L'ESPERIMENTO E' TERMINATO!!! --\n --
L'ESPERIMENTO E' TERMINATO!!! --\n --
L'ESPERIMENTO E' TERMINATO!!! --");
                    else
                        {
                          if
(server.askChange()) {

textArea1.setText(" -- ATTENZIONE! Da ora
affronti nuovi giocatori! --\n -- ATTENZIONE!
```

```
Da ora affronti nuovi giocatori! --\n --
ATTENZIONE! Da ora affronti nuovi giocatori! --
");
                            id1 = id3;
                            id2 = id4;
                        }
                        else

textArea1.setText("Risultati aggiornati, ora
puoi effettuare una nuova scelta.");
                        }
                    try  {
                        server.setBlock3();
                    } catch (Exception ex) {

textArea1.setText(ex.toString());
                    }
                } catch (Exception ex) {

 textArea1.setText(ex.toString());
                }
            }
            else {
                textArea1.setText("Risultati
gia\' aggiornati.");
            }
          }
          else {
            textArea1.setText("Perfavore
aspetta la scelta degli altri\ne quindi invia
nuovamente la tua scelta.");
          }
       }
       else {
           textArea1.setText("Perfavore aspetta
alcuni secondi la scelta\ndegli altri giocatori
e riprova\nad aggiornare i risultati");
       }

     }
 }
    }
}
```

Sum10ModelSwarm.java

```
import swarm.Globals;
import swarm.Selector;
import swarm.defobj.Zone;
import swarm.defobj.SymbolImpl;

import swarm.activity.Activity;
import swarm.activity.ActionGroup;
import swarm.activity.ActionGroupImpl;
```

```
import swarm.activity.Schedule;
import swarm.activity.ScheduleImpl;
import swarm.activity.ActionForEach;

import swarm.collections.List;
import swarm.collections.ListImpl;

import swarm.objectbase.Swarm;
import swarm.objectbase.SwarmImpl;
import swarm.objectbase.VarProbe;
import swarm.objectbase.MessageProbe;
import swarm.objectbase.EmptyProbeMapImpl;

import java.io.*;

/** The Sum10ModelSwarm encapsulates all the
objects used in the
  simulated world itself (but not the user
interface objects) */

public class Sum10ModelSwarm extends SwarmImpl
{

    // simulation parameters
    public int numPlayers, numCycles,
payoffArray[];

    // ActionGroup for holding an ordered
sequence of action
   public ActionGroup modelActions;

    // the single Schedule
    public Schedule modelSchedule;

    // list of producers
    public List playerList;

    // the world of the economic simulation
    //public Market market;
    public Player playerArray[];
    //public Player player2;
    public Timer timer;
    public RuleMaster ruleMaster;
    public RemoteServer remoteServer;
    public Sum10ObserverSwarm
sum10ObserverSwarm;
    public int a, b, c, d, e;
    //public int aA1, aA2, aA3, aA4, bA1, bA2,
bA3, bA4, cA1, cA2, cA3, cA4, dA1, dA2, dA3,
dA4;
    //public int aB1, aB2, aB3, aB4, bB1, bB2,
bB3, bB4, cB1, cB2, cB3, cB4, dB1, dB2, dB3,
dB4;
    // data file
    public String dataFile;

    // file for output datas
    public FileWriter file;
```

```
    // buffered stream for output datas
    public PrintWriter buff;


    // methods to provide access to the objects
inside the model
    public Player[] getPlayerArray () {
  return playerArray;
    }

    // Constructor of Model Swarm
    public Sum10ModelSwarm (Zone aZone,
Sum10ObserverSwarm o) {
  super (aZone);
  sum10ObserverSwarm = o;

  // simulation parameters
  numPlayers = 108;   //prima 27
  numCycles = 35;        //prima 30

  //payoffs
  /**   aA1 = 5;
        aB1 = 2;
        bA1 = 0;
        bB1 = 0;
        cA1 = 0;
        cB1 = 0;
        dA1 = 0;
        dB1 = 0;
        aA2 = 0;
        aB2 = 0;
        bA2 = 3;
        bB2 = 3;
        cA2 = 0;
        cB2 = 0;
        dA2 = 0;
        dB2 = 0;
        aA3 = 0;
        aB3 = 0;
  bA3 = 0;
        bB3 = 0;
        cA3 = 2;
        cB3 = 5;
        dA3 = 0;
        dB3 = 0;
        aA4 = 0;
        aB4 = 0;
        bA4 = 0;
        bB4 = 0;
        cA4 = 0;
        cB4 = 0;
        dA4 = 3;
        dB4 = 3;*/

  /*a = 0;
  b = 1;
  c = 3;
  d = 4;
  e = 10;*/
```

```
  //data storage file
  dataFile = "data/run1.dat";

  //  customized probe map
  class Sum10ModelProbeMap extends
EmptyProbeMapImpl {
     private VarProbe probeVariable (String
name) {
       return

Globals.env.probeLibrary.getProbeForVariable$in
Class
           (name, Sum10ModelSwarm.this.getClass
());
     }
     private MessageProbe probeMessage (String
name) {
       return

Globals.env.probeLibrary.getProbeForMessage$inC
lass
           (name, Sum10ModelSwarm.this.getClass
());
     }
     private void addVar (String name) {
       addProbe (probeVariable (name));
     }
     private void addMessage (String name) {
       addProbe (probeMessage (name));
     }
     public Sum10ModelProbeMap (Zone _aZone,
Class aClass) {
       super (_aZone, aClass);
       addVar ("numPlayers");
       addVar ("numCycles");
       /**addVar ("aA1");
       addVar ("aB1");
       addVar ("bA1");
       addVar ("bB1");
       addVar ("cA1");
       addVar ("cB1");
       addVar ("dA1");
       addVar ("dB1");
       addVar ("aA2");
       addVar ("aB2");
       addVar ("bA2");
       addVar ("bB2");
       addVar ("cA2");
       addVar ("cB2");
       addVar ("dA2");
       addVar ("dB2");
       addVar ("aA3");
       addVar ("aB3");
       addVar ("bA3");
       addVar ("bB3");
       addVar ("cA3");
       addVar ("cB3");
       addVar ("dA3");
```

```
        addVar ("dB3");
        addVar ("aA4");
        addVar ("aB4");
        addVar ("bA4");
        addVar ("bB4");
        addVar ("cA4");
        addVar ("cB4");
        addVar ("dA4");
        addVar ("dB4");*/
        addVar ("a");
        addVar ("b");
        addVar ("c");
        addVar ("d");
        addVar ("e");
        addVar ("dataFile");
        //addVar ("payoffArray[1]");
        //addMessage ("addProducer");
        }
      }

  // installing of custom probeMap class
directly into the
  // probeLibrary
  Globals.env.probeLibrary.setProbeMap$For
  (new Sum10ModelProbeMap (aZone, getClass ()),
getClass ());
     }

     // building of model objects
     public Object buildObjects () {
  int i;


  // allow our parent class to build anything
  super.buildObjects();

  /**payoffArray = new int[4][8];
  payoffArray[0][0] = aA1;
  payoffArray[0][1] = aB1;
  payoffArray[0][2] = bA1;
  payoffArray[0][3] = bB1;
  payoffArray[0][4] = cA1;
  payoffArray[0][5] = cB1;
  payoffArray[0][6] = dA1;
  payoffArray[0][7] = dB1;
  payoffArray[1][0] = aA2;
  payoffArray[1][1] = aB2;
  payoffArray[1][2] = bA2;
  payoffArray[1][3] = bB2;
  payoffArray[1][4] = cA2;
  payoffArray[1][5] = cB2;
  payoffArray[1][6] = dA2;
  payoffArray[1][7] = dB2;
  payoffArray[2][0] = aA3;
  payoffArray[2][1] = aB3;
  payoffArray[2][2] = bA3;
  payoffArray[2][3] = bB3;
  payoffArray[2][4] = cA3;
  payoffArray[2][5] = cB3;
```

```
 payoffArray[2][6] = dA3;
 payoffArray[2][7] = dB3;
 payoffArray[3][0] = aA4;
 payoffArray[3][1] = aB4;
 payoffArray[3][2] = bA4;
 payoffArray[3][3] = bB4;
 payoffArray[3][4] = cA4;
 payoffArray[3][5] = cB4;
 payoffArray[3][6] = dA4;
 payoffArray[3][7] = dB4;*/
 payoffArray = new int[5];
 payoffArray[0] = a;
 payoffArray[1] = b;
 payoffArray[2] = c;
 payoffArray[3] = d;
 payoffArray[4] = e;

 // saving datas
 try {

 // file for output datas
 file = new FileWriter (dataFile);

 // build buffered stream for output datas
 buff = new PrintWriter (file);
 }
 catch (IOException e){
     System.err.println ("Exception in building
data-saving objects: "
                   + e.toString ());
 }
 buff.println ("ID Scelta Punti PuntiTotale
SommaTerna");

 // setting up objects used to represent human
players
 playerList = new ListImpl (getZone ());
 timer = new Timer(numPlayers,
sum10ObserverSwarm, numCycles, buff);
 ruleMaster = new RuleMaster(timer,
numPlayers);
 playerArray = new Player[numPlayers];

 // a loop to create players and put them in
the list
 for (i = 0; i < playerArray.length; i++) {
            playerArray[i] = new Player
(numPlayers, payoffArray, i, playerArray,
timer, ruleMaster);
     System.out.println("Built Player" + i);
     playerList.addLast (playerArray[i]);
 }

 // server class to manage RMI - Remote Method
Invocation
 remoteServer = new RemoteServer(playerArray,
timer, payoffArray);

 return this;
```

```
    }

    // Model Schedule
    public Object buildActions () {
  super.buildActions();

  // create the list of simulation actions. We
put these in
  // an action group, because we want these
actions to be
  // executed in a specific order, but these
steps should
  // take no (simulated) timebuild.
  modelActions = new ActionGroupImpl (getZone
());

  // a step to check number of cycles
  try {
      modelActions.createActionTo$message
        (timer, new Selector (timer.getClass (),
"checkCycle", false));
  } catch (Exception e) {
      System.err.println("Exception in
timer.checkCycle: " + e.getMessage ());
  }

  // main step
  try {
      ActionForEach actionForEach;
      Selector sel =
        new Selector (Class.forName ("Player"),
"step", false);

      actionForEach =
        modelActions.createActionForEach$message
(playerList, sel);
  }
  catch (Exception e) {
      System.err.println("Exception in
player.step: " + e.getMessage ());
  }

  try {
      ActionForEach actionForEach;
      Selector sel =
        new Selector (Class.forName ("Player"),
"step2", false);

      actionForEach =
        modelActions.createActionForEach$message
(playerList, sel);
  }
  catch (Exception e) {
      System.err.println("Exception in
player.step: " + e.getMessage ());
  }

  // a step to make counters for applet-player
synchronization = 0
```

```
  try {
      modelActions.createActionTo$message
        (timer, new Selector (timer.getClass (),
"nullBlock", false));
  } catch (Exception e) {
      System.err.println("Exception in
timer.nullBlock: " + e.getMessage ());
  }

  // a step to make the counter of access to
instructions page = 0
  try {
      ActionForEach actionForEach;
      Selector sel =
        new Selector (Class.forName ("Player"),
"nullInstructions", false);

      actionForEach =
        modelActions.createActionForEach$message
(playerList, sel);
  }
  catch (Exception e) {
      System.err.println("Exception in
player.step: " + e.getMessage ());
  }

  //This schedule has a repeat interval of 1, it
will loop every
  // time step.  The action is executed at time
0 relative to
  // the beginning of the loop.
  // This is a simple schedule, with only three
action that are
  // just repeated every time.
      modelSchedule = new ScheduleImpl (getZone
(), 1);
     modelSchedule.at$createAction (0,
modelActions);

      return this;
     }

     // this model is run as a subswarm of an
observer swarm
     public Activity activateIn (Swarm
swarmContext) {

  // activation of ourselves via the
  // superclass activateIn: method
  super.activateIn (swarmContext);

  // activation of our own schedule
  modelSchedule.activateIn (this);

  // return our activity
  return getActivity ();
     }
}
```

Sum10ObserverSwarm.java

```
import swarm.Globals;
import swarm.Selector;
import swarm.defobj.Zone;

import swarm.activity.Activity;
import swarm.activity.ActionGroup;
import swarm.activity.ActionGroupImpl;
import swarm.activity.Schedule;
import swarm.activity.ScheduleImpl;

import swarm.collections.List;

import swarm.objectbase.Swarm;
import swarm.objectbase.VarProbe;
import swarm.objectbase.MessageProbe;
import swarm.objectbase.EmptyProbeMapImpl;

import swarm.analysis.EZGraph;
import swarm.analysis.EZGraphImpl;

import swarm.simtoolsgui.GUISwarm;
import swarm.simtoolsgui.GUISwarmImpl;

public class Sum10ObserverSwarm extends
GUISwarmImpl {
    // Update frequency of the display
    public int displayFrequency;


    // Action group for sequence of GUI events
    public ActionGroup displayActions;

    // single Schedule of the display
    public Schedule displaySchedule;

    // our swarm
    public Sum10ModelSwarm sum10ModelSwarm;


    // GUI objects:
    // value graph of the first couple
    public EZGraph valueGraph;


    // constructor
    public Sum10ObserverSwarm (Zone aZone) {
  super(aZone);

  // update frequency of the display
  displayFrequency = 1;
```

```
  // Probe Map like in the ModelSwarm
  class Sum10ObserverProbeMap extends
EmptyProbeMapImpl {
     private VarProbe probeVariable (String
name) {
     return

Globals.env.probeLibrary.getProbeForVariable$in
Class
     (name, Sum10ObserverSwarm.this.getClass
());
     }
     private MessageProbe probeMessage (String
name) {
     return

Globals.env.probeLibrary.getProbeForMessage$inC
lass
            (name,
Sum10ObserverSwarm.this.getClass ());
      }
      private void addVar (String name) {
        addProbe (probeVariable (name));
      }
      private void addMessage (String name) {
        addProbe (probeMessage (name));
      }
      public Sum10ObserverProbeMap (Zone _aZone,
Class aClass) {
        super (_aZone, aClass);
        addVar ("displayFrequency");
      }
  }

  Globals.env.probeLibrary.setProbeMap$For
  (new Sum10ObserverProbeMap (aZone, getClass
()), getClass ());
     }

     public Object _valueGraphDeath_ (Object
caller) {
  valueGraph.drop ();
  valueGraph = null;
  return this;
     }

     // build all objects
     public Object buildObjects () {
  super.buildObjects ();

  // build model to observer
  sum10ModelSwarm = new Sum10ModelSwarm (getZone
(), this);

  // build probes in ourselves and in the model
to observ
  Globals.env.createArchivedProbeDisplay
(sum10ModelSwarm,
```

```
"sum10ModelSwarm");

Globals.env.createArchivedProbeDisplay (this,
"sum10ObserverSwarm");

  // build Control Panel and wait for an event
  getControlPanel ().setStateStopped ();

  // let the model build his objects
  sum10ModelSwarm.buildObjects ();

  // Now get the list of players
  Player[] playerArray =
sum10ModelSwarm.getPlayerArray ();

  // The graph of the results of the first
couple (clone it if necessary)

  valueGraph = new EZGraphImpl
      (getZone (),
       "Punteggi",
       "Tempo", "Valore",
       "valueGraph");

  // methods to execute when graphs are closed
  try {

valueGraph.enableDestroyNotification$notificati
onMethod
        (this, new Selector (getClass (),
                             "_valueGraphDeath_",
                             false));
  } catch (Exception e) {
      System.err.println ("Exception in
_valueGraphDeath_: "
                    + e.getMessage ());
  }

  // build values to be rapresented
  try {

valueGraph.createAverageSequence$withFeedFrom$a
ndSelector
        ("Media", sum10ModelSwarm.playerList,
         new Selector (Class.forName ("Player"),
"getPoints",
                      false));
  } catch (Exception e) {
      System.err.println ("Exception in
player.getPoints: "
                    + e.getMessage ());
  }
  try {

valueGraph.createMaxSequence$withFeedFrom$andSe
lector
        ("Max.", sum10ModelSwarm.playerList,
```

```
        new Selector (Class.forName ("Player"),
"getPoints",
                    false));
  } catch (Exception e) {
      System.err.println ("Exception in
player.getPoints: "
                   + e.getMessage ());
  }
  try {

valueGraph.createMinSequence$withFeedFrom$andSe
lector
       ("Min.", sum10ModelSwarm.playerList,
        new Selector (Class.forName ("Player"),
"getPoints",
                    false));
  } catch (Exception e) {
      System.err.println ("Exception in
player.getPoints: "
                   + e.getMessage ());
  }
  return this;
     }

    public Object _update_ ()  {
  if (valueGraph != null)
      valueGraph.step ();
  return this;
     }

    // build actions for the simulation
    // in the Schedule
    public Object buildActions () {
  super.buildActions();

  // build Schedule Model Swarm
  sum10ModelSwarm.buildActions();

  // build Action Group for the display
  displayActions = new ActionGroupImpl
(getZone());

  // put in the Action Group to update the
display
  try {
      displayActions.createActionTo$message
        (this, new Selector (getClass (),
"_update_", false));

      // update of probe display
      displayActions.createActionTo$message
        (Globals.env.probeDisplayManager,
         new Selector
(Globals.env.probeDisplayManager.getClass (),
         "update", true));

      // global update of the GUI
      displayActions.createActionTo$message
        (getActionCache (), new Selector
```

```
            (getActionCache ().getClass (),
"doTkEvents", true));
  } catch (Exception e) {
      System.err.println ("Exception in update
of displayActions : "
                    + e.getMessage ());
  }

  // display Schedule
  displaySchedule = new ScheduleImpl (getZone
(), displayFrequency);
  displaySchedule.at$createAction (0,
displayActions);

  return this;
     }

     //activateIn: activate Schedules
     public Activity activateIn (Swarm
swarmContext) {
  // activate ourselves
  super.activateIn (swarmContext);

  // activate model swarm in ourselves
  // (is an our sub swarm)
  sum10ModelSwarm.activateIn (this);

  // activate and execute Schedule
  // in our selves
  displaySchedule.activateIn (this);
  return getActivity();
     }

     public void drop () {
  if (valueGraph != null)
      valueGraph.disableDestroyNotification ();
  super.drop ();
     }
}
```

Timer.java

```
import swarm.Globals;
import java.io.*;

// Timer class provides synchronization of
human choices and virtual updates

public class Timer {

    // paramethers
    public int numPlayers, cycles, numCycles;
    public int[] block; //expect, expect2;
    public boolean[] round;
```

```
    public Sum10ObserverSwarm
sum10ObserverSwarm;
    public PrintWriter buff;
    //public PrintWriter dataWriter;

    // constructor
    public Timer (int p, Sum10ObserverSwarm o,
int c, PrintWriter b) {
  int i, j, z, x;
  numPlayers = p;
  sum10ObserverSwarm = o;
  numCycles = c;
  block = new int[3];
  for (i = 0; i < block.length; i++) {
            block[i] = 0;
  }
  cycles = 0;
  round = new boolean[numPlayers];
  for (j = 0; j < round.length; j++) {
            round[j] = false;
  }
  buff = b;
  /**expect = new int[4];
  for (z=0; z < expect.length; z++) {
            expect[z] = 0;
  }
  expect2 = new int[4];
  for (x=0; x < expect2.length; x++) {
            expect2[x] = 0;
      }*/
    }

    // methods
    public boolean askChange () {
  boolean ok;
  if ((cycles + 1) == 26) {
      ok = true;
  }
  else  {
      ok = false;
  }
  return ok;
    }
    public boolean askBlock () {
  boolean ok;
  if (block[0] == (numPlayers)) {
      ok = true;
  }
  else  {
      ok = false;
  }
  return ok;
    }
    public boolean askBlock2 () {
  boolean ok;
  if (block[1] == (numPlayers)) {
      ok = true;
  }
  else  {
```

```
        ok = false;
    }
    return ok;
       }
       public boolean askBlock3 () {
    boolean ok;
    if (block[2] == (numPlayers/3)) {
        ok = true;
    }
    else  {
        ok = false;
    }
    return ok;
       }

       public Object checkCycle ()  {
    if (cycles >= numCycles) {
        sum10ObserverSwarm.getControlPanel
  ().setStateStopped ();
    }
    //resetExpect();
    return this;
       }

       public Object nullBlock () {
    while (!(askBlock3())) {
        continue;
    }
    int i;
    for (i = 0; i < round.length; i++) {
              round[i] = false;
    }
    int j;
    for (j = 0; j < block.length; j++) {
              block[j] = 0;
    }
    cycles++;
    buff.flush();
    return this;
       }

       public Object setBlock (int i) {
    block[0]++;
    round[i]=true;
    return this;
       }

       public Object setBlock2 () {
    block[1]++;
    return this;
       }
       public Object setBlock3 () {
    block[2]++;
    return this;
       }

       public boolean getRound(int i) {
    return round[i];
       }
```

```
    public PrintWriter getBuffer() {
  return buff;
    }
    /**    public Object setExpectation(int i)
{
  expect[i-1]++;
  return this;
    }
    public int returnExpect(int i) {
  return ((expect2[i-1]*100)/numPlayers);
    }
    public Object resetExpect() {
  int z;
  for (z=0; z < expect2.length; z++) {
            expect2[z] = expect[z];
  }
  int x;
  for (x=0; x < expect.length; x++) {
            expect[x] = 0;
  }
  return this;
  }*/
}
```

BIBLIOGRAFIA

- Anderson L. e Holt C. (1996), *Information cascades*, in Journal of Economic Perspectives, Volume 10, pp. 187-193

- Akerlof G.A. (1984), *Gift Exchange and Efficiency-Wage Theory: Four Views*, in "The American Economic Review", Volume 74, Issue 2, Papers and Proceedings of the Ninety-Sixth Annual Meeting of the American Economic Association (May), pp 79-83

- Andreoni J. (1995), *Cooperation in Public-Goods Experiments: Kindness or Confusion?*, in "The American Economic Review", Volume 85, Issue 4 (Sep.), pp. 891-904

- Andreoni J. e Miller J.H. (1993), *Rational Cooperation in the Finitely Repeated Prisoner's Dilemma: Experimental Evidence*, in "The Economic Journal", Volume 103, Issue 418 (May), pp. 570-585

- Axelrod R. (1984), *The Evolution of Cooperation*, Basic Books, New York

- Axelrod R. (1981), *The Emergence of Cooperation Among Egoists,* in "American Political Science Review", 75, pp. 306-318

- Axtell R. L. ed Epstein J. M. (1994), *Agent-based modeling: understanding our creations,* in "The bulletin of the Santa Fe Institute", winter.

- Becker G.S. (1976), *Altruism, Egoism, and Genetic Fitness: Economics and Sociobiology*, in "Journal of Economic Literature", Volume 14, Issue 3 (sep), pp. 817-826

- Bergstrom T.C. e Stark O. (1993), *How altruism can prevail in a Evolutionary Environment*, in "The American Economics Rewievs", Volume. 83, Issue 2 (May), pp. 149-155

- Camerer Colin F. (2001), *Behavioral Economics,* Princeton University Press, Princeton

- Cosmides L. e Tooby J. (1992), *Cognitive Adaptations for Social Exchange*, in J.H. Barkow, L. Cosmides e J. Tooby (eds), in "The Adapted Mind", Oxford University Press, New York, pp. 163-228

- Eisenberg N. Miller P.A. (1987), *Empathy, Sympathy, and altruism: Empirical and Conceptual Links*, in "Empathy and its Development", Eisenberg N.e Strayer J. (eds), , Cambridge, Cambridge University Press, pp. 292-316

- Fehr Ernst e Gachter Simon (2000), *"Fairness and Retaliation: The Economics of Reciprocity"*, in "Journal of Economic Perspectives" 14, 3 pp 159-183

- Henrich Joseph, Boyd Robert, Bowles Samuel, Camerer Colin, Feht Ernst, Gintis Herbert e McElreath Richard (maggio 2001), "*Cooperation, Reciprocity and Punishment in Fifteen Small-scale Societies"*, American Economics Review

- Hey John D. (1991), *Experiments in Economics*, Blackwell, Oxford, trad. it. (1998), *Esperimenti in Economia*, G. Giappichelli, Torino

- Holt C. e Laury S. (1997), *Voluntary Provision of a Public Good*, in Journal of Economic Perspectives, Volume 11, pp. 209-215

- Holt C. e Anderson L. (1996), *Understanding Bayes'*, in Journal of Economic Perspectives, Volume 10, pp. 179-187

- Holt C., Grobelnik M. e Prasnikar V. (1999), *Strategic Interaction on the Internet*, in Journal of Economic Perspectives, Volume 13, pp. 211-220

- Holt C. e Sherman R. (1999), *A Market for Lemons*, in Journal of Economic Perspectives, Volume 13, pp. 205-214

- Holt C. e Ball S. (1998), *Speculation and Bubbles in an Asset Market*, in Journal of Economic Perspectives, Volume 12, pp. 207-218

- Holt C. e Laury S. (2000), *Making Money*, Journal of Economic Perspectives, Volume 14, pp. 205-213

- Innocenti A. (1995), *Le origini della experimental economics (1948-1959): una valutazione critica sull'evoluzione delle metodologie sperimentali*, working papers

- Kagel John H. e Roth Alvin E. (1995), *Handbook of Experimental Economics*, Princeton University Press, Princeton

- Kahneman, Knetsch, e Thaler (1986), *Fairness as a Constraint on Profit Seeking: Entitlements in the Market*, in "The American Economic Review", volume 76, Issue 4, (Sep), pp. 728-741

- Kirman A. (1992), *Whom or what does the rapresentative individual represent?*, in "Journal of Economic Perspectives, volume 6, numero 2, pp. 117-136

- Krebs D. (1982), *Psychological Approaches to Altruism: An Evaluation*, Ethics, 92 (3), April pp. 447-58

- Langton C. G. (1992), *Vita artificiale*, in "Sistemi intelligenti", tradotto da Maurizio Riccucci volume 2, pp. 189-246,

- Mark R. Rosenzweig e Kenneth I. Wolpin (Dicembre 2000), *Natural "Natural Experiments" in Economics*, in "Journal of Economic Literature", vol. XXXVIII, pp. 827-874

- Morishima M. (1982), *Why Has Japan 'Succeede'? Western Technology and Japanese Ethos*, Cambridge, Cambridge University Press

- Novarese M. e Rizzello S. (2000), *Economia sperimentale e analisi empirica dei processi di apprendimento*, Laboratorio di Economia Cognitiva - Università del Piemonte Orientale, Alessandria, working paper

- Ochs J. (1995), *Coordination Problems*, in J.H. Kagel & A.E. Roth (eds) The Handbook of Experimental Economics, Princeton University Press, Princeton, pp. 195-251

- Parisi D. (2001), *Simulazioni, la realtà rifatta al computer*, il Mulino, Bologna

- Rabin M. (1993), *Incorporating Fairness into Game Theory and Economics*, in "The American Economic Review", Volume 83, Issue 5 (Dec), pp. 1281-1302

- Rotemberg J.J. (1994), *Human Relations in the Workplace*, in "The Journal of Political Economy", Volume 102, Issue 4 (Aug), pp. 684-717

- Roth Alvin, Prasnikan Vesna, Okuno-Fujiwara Masahiro, Zamir Shmuel (1991), *"Bargaining and Market Behavior in Jerusalem, Ljublliana, Pittsburgh and Tokyo: An Experimental Study",* in "American Economic Rreview"

- Sacco P.L. e Zamagni S. (1994), *Un'approccio dinamico evolutivo all'altruismo*, in "Rivista Internazionale di Scienze Sociali", 4-6, CII:2, pp. 223-262

- Samuelson P. e Nordhaus W.D. (1985), *Principles of Economics*, XII ed., McGraw-Hill, New York

- Shafir E. e Tversky A. (1992), *Thinking through uncertainty: Nonconsequential reasoning and choiche*, Cognitive Psychology, 24, pp. 449-74

- Simon H.A. (1983), *Reason in Human Affairs*, Stanford, CA, Stanford University Press, trad. it. Simon (1984), *La ragione nelle vicende umane*, Il Mulino, Bologna

- Simon H.A. (1993), *Altruism and Economics*, in "The American Economics Rewievs", Volume 83, Issue 2 (May), pp. 156-161

- Skyrms B. (1996), *The Evolution of Social Contract*, Cambridge, Cambridge University Press

- Trivers L. (1971), *The evolution of reciprocal altruism*, Quarterly Review of Biology, 46, pp. 35-57

- Tullock G. (1985), *Adam Smith and the Prisoners' Dilemma*, in "The Quarterly Journal of Economics", Volume 100, Issue Supplement, pp. 1073-1081

- Tullock G. (1999), *Non-prisoner's dilemma*, in "Journal of Economic Behavior & Organization", Vol. 39, pp. 455-458

- Williams G.C. (1966), *Adaptation and Natural Selection: A Critique of some current evolutionary thought*, Princeton University Press, Princeton