人工社会における権力システムの創発に関する研究

課題番号　１２８３４００４

平成１２年度～平成１３年度科学研究費補助金

（基盤研究（Ｃ）（２））

平成１４年　３月

研究代表者　高木英至

（埼玉大学教養学部教授）

はしがき

本報告書は「人工社会における権力システムの創発に関する研究（課題番号　１２８３４００４)」という課題で得た科学研究費に基づく研究成果をまとめる。

この研究課題にはポイントが２つある。第１は、同課題の目的が計算機内でシミュレートした「人工社会(Artificial Society)」の中で仮想のエージェントがどのような関係性を形作るかを調べることにあることである。すなわち、多くの分野で普及しつつある計算手法、特に進化的計算手法を社会科学の領域で適用することを目的としている。進化的計算手法が社会科学のモデル構成法としてどれほど有望であるかを見極めることが、この研究課題の背後にある関心である。

第２に、同研究課題は、社会秩序のうちの「権力(Social Power)」という側面を取り組む対象として定めている。社会は協力性、階層性、集団標識ごとの分離、など様々な社会秩序特性で特徴づけられるけれども、「他者に自分の望む行動をとらせる能力」としての権力の存在は、社会秩序を特徴づける、たぶん最も重要な特性の１つである。

ただし同課題の２年間の研究期間で解明できた範囲は自ずと限定される。その限定は、１つには権力システムの創発をモデル化する以前に、別の形式の社会過程のモデル化が必要であったことにも起因している。例えば研究代表者のモデルでは、権力は一定の交換から生じることを想定している。そこで社会的交換のシミュレーションなどを準備段階として行うことを余儀なくされた。従って同課題の下で行った研究がすべて、直接権力のシミュレーションそのものであった訳ではない。

目次にあるように、本報告書は４つの部に分かれている。

第１部（Ｉ）は、研究代表者が想定した、社会的交換から権力関係が生じる、というモデルを計算実験で確かめた研究を収録している。次の３つの論考から構成されている。

Ｉ．１．権力発生の計算モデル
（高木英至　(2000)　「権力発生の計算モデル」、『埼玉大学紀要』、第 36 巻、第２号、２３－３７頁）

Ｉ．２．A computational model of the emergence of social power
(Takagi, E. Paper presented at the 9th International Conference on Social Dilemmas, at Chicago, 2001.)

Ｉ．３．社会的交換に基づく権力の発生
（高木英至　(2001)　「社会的交換に基づく権力の発生」、『日本社会心理学会第 41 回大会発表論文集』）

Ｉ．１．は社会的交換の偏りから権力が生じると考える、最初に構成した計算モデルに

基づくシミュレーション結果をまとめている。ただしこの第１のモデルには多くの問題があった。問題のいくつかを回避した第２のモデルに基づいたシミュレーション結果をまとめたのがⅠ．２．とⅠ．３．である。

第２部（Ⅱ）は、エージェントの（いわば）空間的配置に基づく、多数派　対　少数派の権力関係を問題にした論考である。

Ⅱ．１．Social Impact シミュレーションのタネと仕掛け
（高木英至（2000）「Social Impact シミュレーションのタネと仕掛け」、『日本グループ・ダイナミックス学会　第４８回大会　発表論文集』、pp.62-63.）
Ⅱ．２．相互調整によるエージェントのクラスタ化：コンピュータシミュレーションによる検討
（次の学会報告はこの論考の短縮版である：高木英至（2000）「相互調整によるエージェントのクラスタ化：コンピュータシミュレーションによる検討」『日本シミュレーション＆ゲーミング学第12回全国大会発表論文集』、pp.10-13.）

Ⅱ．１．は Nowak & Latané らの Social Impact Theory に基づくシミュレーションモデルの格調モデルに基づいた分析である。Ⅱ．２．は上記のモデルの前提を変えて考案した、より一般的な「相互調整モデル」に基づいたシミュレーションによる分析である。

第３部（Ⅲ）は権力のモデルを構築する上で前提となった社会的交換、ないし協力性の創発について行った分析結果をまとめている。以下の論考からなる。

Ⅲ．１．文化の成立に関する試論：交換規範はいかに創発するか
（高木英至（2001）「文化の成立に関する試論：交換規範はいかに創発するか」、『埼玉大学紀要』、３７巻、２号（印刷中）.）
Ⅲ．２．利他性
（高木英至（2002）「利他性」、『月刊　言語』、３１巻、２号、１２－１５頁）
Ⅲ．３．協力呼びかけのシミュレーション：安心は信頼を低下させる

Ⅲ．１．は「限定交換」という種類の社会的交換のシミュレーション分析である。第１部で用いた計算モデルは、権力は社会的交換の偏りから生じるというアイディアに基づいていた。その意味で、同モデルは権力の創発メカニズムが交換メカニズムに依存すると考えている。しかし第１部の分析をする上での制約は、交換自体のモデル化に不明の点があったことである。特に第１部の分析は、交換においてエージェントは（与えるとすれば）必ず定量の資源を相手に与えると仮定した。この仮定をゆるめ、可変量の資源を与えるこ

とを許す（量自体は戦略が決める）ときにいかなる効果が生じるかを推論することが、Ⅲ．１．の目的である。

だがⅢ．１．の分析結果は、当初の思惑以上の大きな意味があったと、現在の時点で筆者は考えている。Ⅲ．１．は結果として、「互酬規範（Reciprocity Norm）」という人類に普遍的な文化要素が、進化ゲームの論理で生じ得ることを示しているからである。Ⅲ．１．は特に、「文化要素の創発」という視点から交換関係を論じることとなった。

Ⅲ．２．は進化行動学に関する連載の一貫として筆者が執筆した原稿の再録である。この論考の素材は主として筆者の以前の研究結果に基づいている。ただし、利他性ないし協力性は権力の背景となる現象であり、同時にⅢ．１．と関連の深いことがらである。

Ⅲ．３．は交換ではなく、あるエージェントが順番に、他者に協力を呼びかけて協力関係を形成する状況をシミュレートしたモデルに基づいている。権力関係も、あるい意味では協力の呼びかけによって成り立つ。そこで協力を呼びかけた場合の協力関係の成否のメカニズムを、特に選択的受け入れ（selective inclusion）という視点からモデル化している。このシミュレーションの結果の含意は２つある。第１は、潜在的には社会的ディレンマ状況である協力関係も、選択的受け入れメカニズムをエージェントが戦略進化させるために、容易に可能になる、という点である。第２は、協力関係への違反への誘因が存在する場合にはしない場合よりかえって、信頼感（trust）、つまり未知の他者に対する協力の予期が高まる、という点である。この結果もⅢ．１．の結果と同様に、当初の想定を越えた大きな成果であったと考えている。

第４部（Ⅳ）にはⅠ～Ⅲの研究（Ⅲ．２．は除く）で用いたシミュレーションプログラムのソースコード（Pascal 部のみ）を掲載した。

研究代表者　高木英至

２００２年　３月

研究組織

研究代表者　　高木英至　（埼玉大学教養学部教授）

研究経費

平成１２年度　　９００千円

平成１３年度　　４００千円

研究発表

(1) 学術誌等

高木英至 (2000) 「権力発生の計算モデル」、『埼玉大学紀要』、第 36 巻、第 2 号、２３－３７頁

高木英至 (2001) 「文化の成立に関する試論：交換規範はいかに創発するか」、『埼玉大学紀要』、３７巻、２号（印刷中）.

高木英至 (2002) 「利他性」、『月刊　言語』、３１巻、２号、１２－１５頁

(2) 口頭発表

高木英至 (2000) 「Social Impact シミュレーションのタネと仕掛け」、『日本グループ・ダイナミックス学会　第４８回大会　発表論文集』、pp.62-63.

高木英至 (2000) 「相互調整によるエージェントのクラスタ化：コンピュータシミュレーションによる検討」　『日本シミュレーション＆ゲーミング学第 12 回全国大会発表論文集』、pp.10-13.

高木英至・工藤恵理子・神信人 (2000) 「社会心理学におけるコンピュータシミュレーションの利用」、『日本社会心理学会第 41 回大会発表論文集』、p.15.

Takagi, E. (2001) Paper presented at the 9th International Conference on Social Dilemmas, at Chicago.

高木英至 (2001) 「交換規範の成立：分配規範と互酬規範」　『日本シミュレーション＆ゲーミング学第 13 回全国大会発表論文集』.

高木英至 (2001) 「社会的交換に基づく権力の発生」、『日本社会心理学会第 42 回大会発表論文集』.

高木英至 (2001) 「「社会的事実」はなぜ存在できるか？－適応／進化の視点から社会構造を考える」、人間進化行動学会全国大会講演（東京大学、駒場）

(3) 出版物

なし

目次

Ⅰ．1．権力発生の計算モデル

高木英至（埼玉大学　教養学部）

【要約】本稿は権力発生の一経路を計算モデルによって検討する．他者に罰を科せる条件下で仮想のエージェントに協力の交換をさせるとき，交換から生じる「強さ」の偏りが交換の流れに作用し，協力を過大受領する「権力者」を発生させる．このアイディアが成り立ち得ることを計算実験で示すとともに，同モデルの問題点を議論する．(1)

1．計算モデルによる権力発生の検討

権力ないし社会的勢力は社会科学の最重要概念の１つである．社会には普遍的に権力が存在する．社会や集団の構造はほとんど常に，権力構造として特徴づけることができる．しかしその権力がなぜ存在し得るか，どのように発生するかについては，理論的にはあいまいである．　本稿が問題にするのは，権力の発生を説明する計算モデルの可能性である．

1．1．理論としての計算モデル

理論ないし説明モデルには３種類がある(Ostrom, 1988)．第１は自然言語で記述した言語モデル，第２は数理モデル，第３はコンピュータのコードで記述された計算モデル（computational models）ないしコンピュータシミュレーションモデルである．

社会科学を含め「文系」の領域では元来，理論（モデル）は言語モデルとして記述されて来た．経済学者を除けば，社会科学の理論家はややもすれば，文献解読や文献解読に基づく思索を生業としてきた．データ分析に計量モデルを用いるとしても，理論自体は言語モデルとして表現することが多かった．経済学を別にして，社会科学における数理モデルの導入は限定的だった．社会学や政治学で数理モデルとして成功したのは，大半が（ゲーム理論を含めた）経済学の手法を導入した領域だった．

社会科学が言語モデルを用いるのはむろん自然な流れである．その理由には少なくとも次の２つがあるだろう．第１に，社会科学の「理論」がしばしば，前提から一定の帰結を導くような本来の理論ではなく，現実を記述するカテゴリー（群）に過ぎなかったことである．カテゴリーを自然言語以外で表現することには固有の意味はほとんどなかったはずである．第２に，社会科学がその議論の基礎とする人間行動のメカニズムが多分に「質的」であり，通常の数学的概念による表現になじみにくかった点をあげることができる．

しかし言語モデルには厳密性において欠点がある．第１に，モデルの論理的一貫性を確認することが言語モデルは不得意なことである．自然言語の操作は論理的矛盾を自ずと導

くものではない．この点はモデルが複雑になるほど妥当する．第２は，言語モデルでは背後の暗黙の仮定が隠れたままになりやすいことである．第３に，モデルがどれほどの含意（implication）を持つかを評価することも難しい．われわれはモデルが導きたい帰結や含意を導けたなら，それ以上の操作をしないだろう．しかしモデルは，説明したい効果とともに説明したくない効果（つまり妥当性が期待できない結論）を含意しているかも知れない．

これらの言語モデルの欠陥に対処できるのは数理モデルと計算モデルである．両者ともモデル内の論理的矛盾を識別しやすく，モデル構成者に仮定を明示することを強制し，いろんな側面の帰結を導くことを得意としている．

ただし両者は同じ性格の存在ではない．計算モデルには数理モデルに明らかに劣る点がある．数理モデルが一定の結論を証明するものであるのに対し，計算モデルは証明をしない．sensitivity analysis はこの問題に対処する有力な方法である．が，原則として計算モデルにできることは「例を出す」ことだけである．つまり計算モデルは数理モデルに比べて厳密性，一般性に限界を持っている．

しかし計算モデルには数理モデルにはない利点もある（e．g．, Taber & Timpone, 1996）．筆者が考える利点の第１はモデル構成の容易さである．計算モデルは要素的なステップの集積からなっており，個々のステップの構成は比較的やさしい．第２は特に，言語的アイディアをモデルに移植することが容易な点である．例えば個々のステップは if ～ then ルールなど，自然言語の発想に近い演算で構成することができる．また，計算機向けのビット列コーディングなどで「質的」な属性を表現することに優れることである．特に人間行動をモデル化する場合はこの点が大きな助けになる．第３はモデルの融通性，柔軟性である．例えばいったん構成したモデルの改変や条件追加などは一般に容易である．

計算モデルはその限界にもかかわらず利点によって，多くの利用者を獲得しつつある．

１．２．人工社会の課題

社会科学における計算モデルは通常，次のように構成される．複数のエージェント（行為者）を仮定し，そのエージェントの行動ルールなどローカルなルールを前提としてエージェント間の相互作用を導くようなモデルである．エージェント間の相互作用の総体は計算機の中の仮想の社会，すなわち人工社会（artificial societies）と呼ぶことができる（Epstein & Axtell, 1996）．人工社会に基づく研究の課題は，ローカル／マイクロなルール（エージェントのレヴェルのルール）から社会のグローバル／マクロなルール（社会構造，社会秩序）を導出することにある．

社会科学の計算モデルの中に「進化型のモデル」と呼び得る形式がある． 進化型のモデルとは進化ゲームの形式を備えた計算モデルを指す．すなわち，エージェントの行動ルール（しばしば戦略，遺伝子と呼ぶ）が相互作用のエージェントに対する結果に応じて変化し，エージェント間の戦略分布が変化（進化）する，というモデルである．進化型のモデ

ルは結果に応じてエージェントの行動ルールを変化させるルール（進化ルール）を組み込んでいる．進化は生殖再生産に基づいて生じてもよい（良い結果を得たエージェントが自己の遺伝子をコピィした子孫を多く残し，良い結果を残せなかったエージェントの遺伝子は淘汰される）．あるいは，観察学習によって生じてもよい（悪い結果を得たエージェントが良い結果を得たエージェントの戦略を学習する）．進化型のモデル以外は便宜のため，「単純推論型のモデル」と呼んでおこう．

注意すべきは，エージェントの行動ルールの組が社会の状態を表現する点である． 行動ルールの組とは，エージェントｉの行動ルールをａｉとしたときの，

ａ＝（ａ１，ａ２，・・・，ａｎ），　ただしｎはエージェント数

である．あるいは，行動ルールの組におけるある規則性が，社会構造ないし社会秩序を表している．

進化型のモデルが単純推論型と異なるのは，単純推論型では与件として固定されるエージェントの行動ルールや社会の構造が内生変数となっていることである．このことは，進化型モデルから生み出される人工社会が「自己組織的」な性格を帯びていることを意味する． 同時に，いい換えれば，シミュレーションのパラメータ（の少なくとも一部）をモデル自体の作動に任せることを意味している．

進化型のモデルは社会の状態を進化的均衡，ないし動的均衡(Thomas, 1984)として説明しようとする．つまり原則としてエージェントは自由に自らの存在（行動ルール）を変えられると仮定した上で，エージェントの存在したがって社会の状態が行き着く均衡を求めようとする．進化型モデルから予測される均衡とは，マイクロなルールから生まれるマクロレヴェルの「創発性」ということができる．

進化型モデルの固有の存在意義は，与件を自己生成するという上記の特性の中にある．経験的にわれわれは，人間（ヒト）が固有の行動特性を持ち，あるいは社会が文化的な差異にかかわらず共通の構造，秩序を持つことを観察し，その観察事実を所与と考える．だがそう考えただけでは，当の行動特性や構造がなぜ社会に存在するかを説明する課題は達成できない．進化型モデルに基づくシミュレーションの意義は，そうした与件の成立根拠を推論することに求められる．

１．３．権力の理論

権力の存在は社会構造の普遍的特性の重要な１つである．むろん「中央政府」にあたるような権力がない社会はしばしば観察されている(Sigrist, 1967)．しかし何らかの支配序列を持つことは，霊長類の社会の「文法」の１つといえる(Haslam, 1997)．進化型の計算モデルが既述のように社会の与件を説明するものであるならば，権力という普遍的な社会構造特性の存在が進化型のモデルに委ねるべきことは自然な発想といえるだろう．

私見では，従来の権力の理論は権力の要因論とでも呼ぶべき理論だった．その典型がFrench & Raven (1959)の権力基盤 (Bases of Power) の議論である．この理論（というよ

り図式）は権力（勢力）の基盤として強制力，報酬，正当性，専門性，参照性，情報をあげ，それらの基盤に応じて対人的な勢力の強さが決まることを整理している．ある意味では M.Weber（1922/1947）の権力図式も，正当性の根拠に応じて権力を分類しているという点では，French らの図式と類似している．

従来の権力論の中で簡潔で強力な視点を提示するのが Emerson（1974a，b）の権力－依存（Power-Dependence） パラダイムである．この権力理論は，社会的交換において行為者Ａが行為者Ｂに利得を依存するほど，Ａに対するＢの権力が強まることを述べる．Emerson の理論も，依存という権力を生む要因（French らの強制基盤と報酬基盤にあたる）を示したものであるため，権力の要因論の１つといえる．が，Emerson の議論が他の交換理論の権力論（e.g.，Blau，1964）より優れているのは，いち早く交換ネットワークの概念を明示し，交換ネットワークという構造的要因が権力を生み出していることを明示したことである．

しかし進化型のモデルを標榜する立場からは，以上のような権力の要因論は満足できるものではない．権力の存在を創発的に説明する訳ではないからである．権力の要因論が述べるのは，特定の行為者に要因や基盤が集まったときにその行為者に権力が生まれることである．しかしこの議論自体は何れかの行為者にその要因や基盤が集中することを説明するものではない．別のいい方をすれば，特定の行為者に基盤が集まることが動的均衡であることを述べる訳ではない．進化型のモデルによって権力の成立を説明するとは，まさに動的均衡として何らかのエージェントに権力（基盤）が集まることの創発的説明をすることに他ならない．その創発的説明が目指すのは，権力を内蔵した社会秩序がなぜ普遍的に生まれるのか，という問に答えることである．

権力が創発的に生じる経路はいくつかあるだろう．１つ考えられるのは，集団の機能として権力が生成される場合である．コミュニケーションネットワーク実験の集団のように，成員が目標を共有して協同する場合を考えてみよう．条件によっては，中心的な位置にある特定の１人に情報を集め，その１人が他の成員に指示を出すような体制が自動的に生まれる可能性がある(高木，1999)．このとき中心的な成員には集団機能としての権力が備わるだろう．また，より広い社会において，ある便益的な社会秩序の成立とともに特定の成員に権力が生じる可能性があるかも知れない．

集団機能として権力が生じる経路の中の重要な可能性は，各エージェントが特定のエージェントに「権限の委譲」をすることである．共有資源の維持・管理といった，個人的対処では解決できない問題に直面したとき，人間集団は「代表」に権限を委譲することは，実験的研究において知られている(Messick et al.，1983)．さらに，Suleiman & Fischer (2000) は，集団間対立があるときに集団の代表者への権限委譲の様式によって集団間関係がいかに変容するかをコンピュータシミュレーションで検討している．権限委譲の仕方が集団間関係に影響を与えることは，逆にいえば集団間秩序の形成に応じて権限委譲の様式も創発されることを示唆している．

が，本研究が扱うのは，集団機能として権力が生じるという経路ではない．エージェント間の自発的な社会的交換の偏りから権力が形成される可能性である[2]．

１．４．社会的交換による権力発生の説明

本稿が想定するのは交換ネットワークから権力が成立する経路である．

筆者がここで念頭に置くのは小説やコミックに描かれる劉邦（後の漢の高祖）や三国志の劉備の姿である．歴史上の劉邦から離れて，次のような架空の劉邦像を描いてみよう．まず劉邦は半ば侠客集団の中心人物として出発する．その時点での劉邦は仲間の世話をし，同時に仲間から協力を得る存在であった．つまり仲間の間での社会的交換が，劉邦を中心に発達していた（図１，（ａ））．しかしいったん劉邦が仲間から協力を安定的に得るようになると，他者には劉邦が仲間の協力を得る存在として映るようになる．仲間やその他の人から見れば，劉邦は個人以上の力を持つ存在となる．つまりもし逆らえば，自らに大きな被害を与える存在となる．このような過程を経て，劉邦は仲間の１人以上の存在となって行く．もはや他者の協力（あるいは服従）を得るのにその人に世話を与える必要はない（図１，（ｂ））．強い権力を確立するのである．

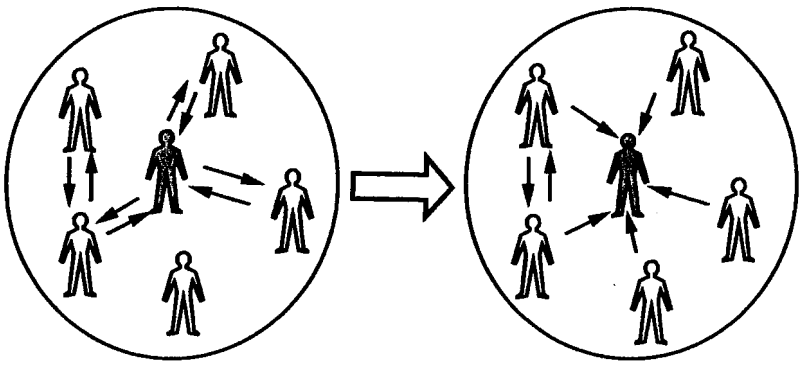


図１：社会的交換から権力へ

以上のアイディアをより公式に表現すれば次のごとくだろう（図２）．人は相互になんらかの協力（労力）のやりとり（社会的交換）をする．さらに，人には多かれ少なかれ，相互の強弱に応じて他者を罰する能力（科罰能力）を持つと仮定する．しかし社会の中では労力の社会的交換において偏りができる．つまり交換に成功しより多くの交換に参加する者と，孤立する者とが生まれる．多くの交換に参加し他者の協力を得ることができた者はその交換相手から得る協力の分だけ強さを増し，科罰能力を高めるだろう．さらに，社会の中では誰が強い（科罰能力がある）かについての評判（reputation）ができあがるだろう．いったん「強い」と認知された者は，その科罰能力のために，自らは協力を供出することなく協力が得られるようになる．その結果得られる協力がさらに当人の権力を高める．このサイクルは適当な均衡点に達するまで続くことになるだろう．

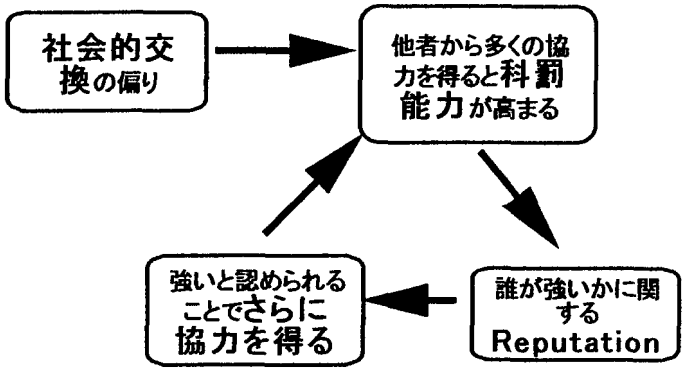


図2：権力発生のサイクル

このように，本稿で想定するのは，権力の保持が権力の基盤になるという，権力の再帰的性格(recursivity)である．すなわち，社会的交換を経て他者の協力によって権力を持つというその事実が他者の服従をさらに促進し，権力を補強する，という点である．

ただし，以上のアイディアはあくまで言語モデルである．言葉の上では成り立つような気がする．が，ここで想定するような「誰かが権力を得る」という状態は，動的な均衡として整合的に矛盾なく到達できるのか？ ― この点は言語モデルだけでは確定できない．以下では「交換から再帰的に権力が生まれる」というアイディアの論理的整合性を検討することを目的として計算モデルを作り，コンピュータシミュレーションによって権力の発生を検討しようとする．

2．シミュレーションモデル

2．1．モデルの概要

本稿が報告するシミュレーションモデルはある２次元空間内にエージェントが並んだ社会を考える．

１５×１５（＝２２５）のセルからなる空間を考える．各セルには仮想のエージェントがいる．エージェントは近隣の２４のエージェントだけと接触できる．エージェントの行動は保持する戦略によって決まる．戦略は交換戦略と科罰戦略の組である．エージェントは各試行で最初に交換局面に入り，次に科罰局面に入る．ある試行でのエージェントの利得とは，交換局面で得た利益と科罰局面で得た負の利得の和である．まず交換局面で，エージェントは他のエージェントに自分の資源（協力）を与えることができる．他者からの資源はエージェントの利益になる．また，資源を与えることは次の科罰局面で相手に協力の約束することを意味する．科罰局面でエージェントは他者に罰を与えることができる．エージェントは「強さ」の値を持ち，科罰できるのは相手が弱い場合だけである．さらに，

同じ試行の交換局面で他者から資源をもらったとき，その他者の強さの半分の値がエージェントの強さに加わる．したがって多くの資源を受けたエージェントは強い．罰のコストは科罰エージェントとその協力者で，罰の被害は被科罰エージェントとその協力者で，それぞれ分担する．シミュレーションの１ラウンドはこうした試行の２００回からなる．ラウンドの各エージェントの得点は各試行での利得の割引加重和である．次のラウンドに移るときには，利得の高かったエージェントの戦略が低かったエージェントの戦略と入れ替わる（学習）．ラウンド終了時に一定の確率で戦略の突然変異をおこす．

３条件でシミュレーションを行う．科罰なし条件は統制条件であり，エージェントは罰を科することなく交換だけを行う．科罰－平等条件では各エージェントは等しく値２の強さを持つ．科罰－不平等条件では３割のエージェントが強さ３，残りが強さ１の値を持つ．条件ごとに１０の Run を行う．各 Run でラウンドは１００までである．

２．２．空間特性

セル空間は上下左右がつながった torus 状の形状をなす．torus であるためどのセルも「中心－周辺」の点では同じである．セル間の距離をブロック距離で定義する．したがって近傍の定義は von Neuman 近傍（最小距離で隣接するのは上下左右の４セルのみ）である．図３で例示すれば，特定のセル（黒いセル）に隣接する８つのセルのうち上下左右の４つのセルが距離１であり，斜めに隣接するセルとの距離は２である．各エージェントは距離３までの計２４のエージェント相互作用することができると仮定する．エージェントの移動はないと考える．

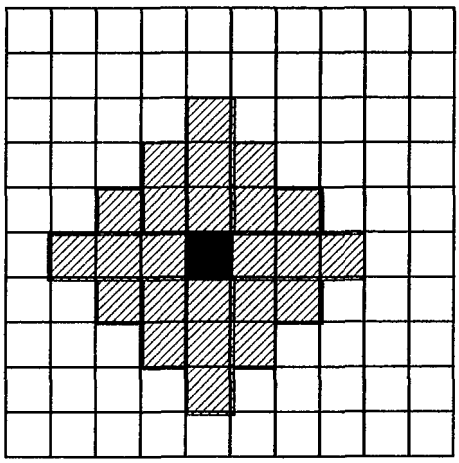


図３：セル空間
（斜線部は接触可能範囲）

２．３．交換と科罰の戦略

エージェントの戦略は１６の２進数（０１）の組からなる（図４）．最初の１０の２進数

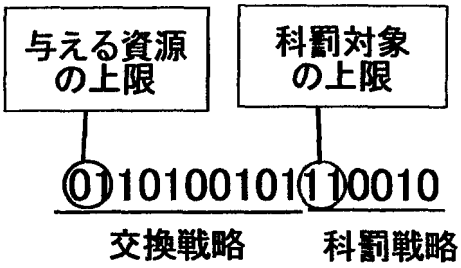


図4：エージェントの戦略の例

が交換戦略を，後の６が科罰戦略を表す．

交換戦略のうち２つの２進数は他者に与える（交換に供する）資源量の上限を表す．１０進数に変換したときの０～３に４をかけた値が交換資源の上限である．１試行で相手に与える資源量を１に固定する．１試行で資源を与え得る相手は２４の他者のうち１２である．手元に残した資源は自分で保有する．自分の１資源は利得１，他者から得た資源は利得２と換算する．したがって与えるか否かの判断はＰＤの性格を帯びる．残りの８つの２進数は資源を与える相手を誰にするかを定義する．エージェントは相互作用できる他者を次の３次元で８分類する：前試行で自分より強かったか否か（２），前試行で自分に資源を与えたか否か（２），前試行で自分に罰を与えたか否か（２）．この８分類（２×２×２）に対応する２進数の値が１なら，その分類の相手を与える候補に含める．エージェントは，相互作用が可能な他者の中からこの戦略の基準に合う候補者を選ぶ．そして候補者の中から交換資源の上限まで，実際に与える相手をランダムに選択する．

科罰戦略の最初の２進数は科罰対象の上限を表す．同様に０～３に３をかけた数だけ，各エージェントは科罰対象者を選ぶことができる．エージェントは他者を次の２次元で分類する：前試行で自分に資源を与えたか否か（２），前試行で自分に罰を与えたか否か（２）．実際の科罰対象者を選ぶ手順は交換戦略と同様である．ただし科罰は弱い相手に対してしか生じない．

３条件を通してエージェントは同じ１６次元の戦略によって動く．ただし科罰なし条件では科罰戦略は値にかかわらず作動せず，相手からの科罰も常に「なし」である．つまり科罰なし条件では，戦略のうち交換戦略の６次元だけが有効となる．

２．４．強さと科罰

ある試行の交換局面でＡがＢに資源を与えたとき，その資源は労力だと考え，同試行の科罰局面ではＡはＢに協力すると仮定する．つまりＡの強さの半分がＢに加わる．ただしＡがＣにも資源を与えていれば，Ｃに対するＢの強さの計算ではＡの強さは加算しない．つまりエージェントは協力者間の喧嘩には介入しないと仮定する．そのため，エージェントの利用可能な資源は相手によって変わることになる．エージェントの強さはその相手ごとに計算し直して定義する．

科罰側には科罰コスト（-0.5），被罰側には被害（-3 - 強さの差）が生じる．コスト，被害の半分はそれぞれ，科罰エージェントと被科罰エージェントが負担する．残りの半分はそれぞれの協力者が等しく分担すると仮定する．この仮定は，弱いエージェントに協力すると科罰によるコストを抱え込むことを意味している．

２．５．その他の設定

各 Run の開始時にエージェントの強さ，戦略の値を乱数で決める．戦略の各値が０か１かは確率 0.5 である．科罰不平等条件では，エージェントは確率 0.3 で「強い」エージェントになる．強さは Run を通して変わらない．

あるラウンドでのエージェント i の利得を X_i，そのラウンドの試行mでのそのエージェントの利得を x^i_m とするとき，

$$X_i = \sum_{m=1}^{n} x^i_m \cdot w^{m-1}$$

である．１ラウンドを２００試行と仮定したのでn= 200，割引重みづけは w = .975 である．

各ラウンドの終了時に，利得が下位１／１５のエージェントの戦略を，上位１／１５の中からランダムに選んだ戦略に取り換える．科罰不平等条件の場合，戦略の入れ替えは強さ集団（２水準）ごとに行う．各ラウンドの終了時に，各エージェントの戦略の２進数の各々は確率 .005 で異なった値に変化する（突然変異）．

３．結果

３．１．交換と科罰の概況

１００ラウンドを５つのラウンドブロックに分けて，１エージェントが１試行当りで与えた資源量，罰度数，利得を分析した．

与える量についてはラウンドブロックの主効果だけが有意だった(図５，$F(4,108)=99$, $p<.001$)．第２ブロックで与える量が増える．条件間の差はなかった．エージェント間での与える量のバラツキ（標準偏差）でも条件間の差はない．最終ブロックでの与えた量の平均は科罰なし(6.3)，科罰－平等(6.8)，科罰－不平等(6.9)だった．つまり平均して，最大資源量（12）の半分強を他者に与えるのに使ったことになる．

罰の度数でもラウンドブロックの主効果だけが有意だった($p<.05$)．科罰は後半に増加する．条件間では差がない．最終ブロックでの平均度数は 1.80(科罰－平等)と 1.73（科罰－不平等）である．

当然ながら利得の平均値は，科罰のコストと被害がある両科罰条件で低い．

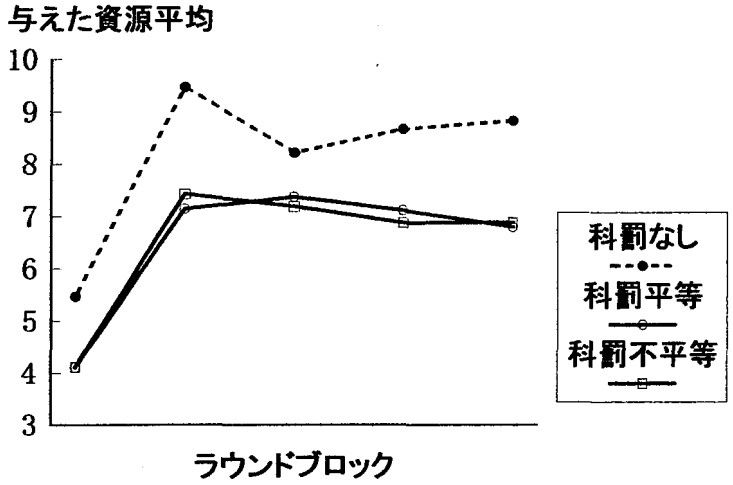


図5：与えた資源量の平均値の推移

3．2．授受のバランス

与える量に条件間の差がなくても，与える量と受取る量の差額では条件間の差は大きい．図6はラウンドブロックを通しての「受取った平均量－与えた平均量」の標準偏差を表す．2つの科罰条件では差額のバラツキが大きい．つまり多くを受取ってあまり与えないエージェントとその逆のエージェントが出現していることになる．

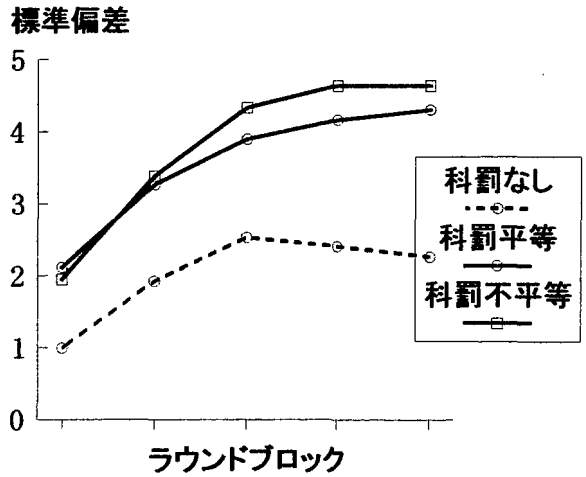


図6：授受差額の標準偏差

3．3．利益の相互性

ラウンドブロックごとにエージェントが受取った量と与えた量の相関係数をとった結果

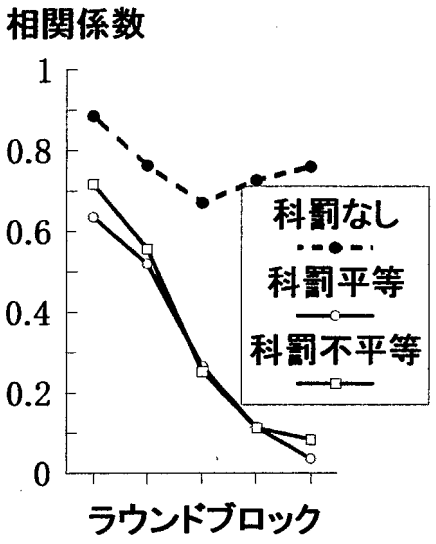
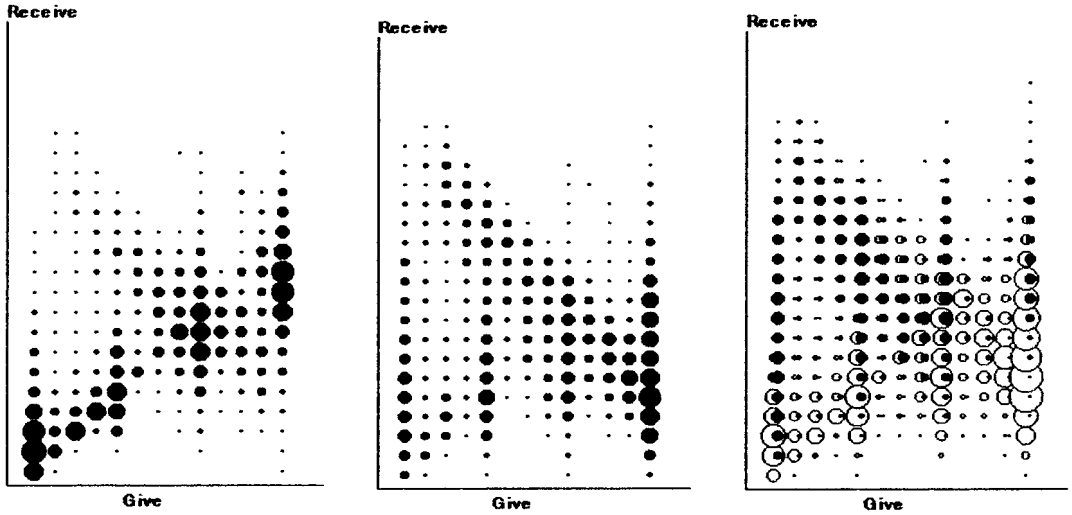


図7：与えた量と受取った量の相関係数

を表すのが図７である．相関係数が 1.0 であればエージェント間で完璧な相互性(reciprocity)が成り立ったことになる．科罰なし条件では相関係数が終始高い値を維持するのに対し，両科罰条件ではラウンドとともに相関が低下しゼロに近づいている．この結果も３．２．の結果同様に，科罰のある条件では授受の不均衡があるエージェントが出現している（そのために相関係数が低下した）ことを表している．

図８は条件ごとに，エージェントが与えた資源量と受取った資源量との最終ブロックでの関連を示す．丸の大きさ（面積）が特定の与えた－受取った資源量の組に該当するエー



科罰なし条件　　科罰－平等条件　　科罰－不平等条件

図8：授受資源量の関連

ジェント数を表している．図８の結果も図７の結果を反映している．科罰なし条件では与えた量と受け取った量はほぼ相関している．多く与えたエージェントは多く受取っている．しかし科罰－平等条件ではこの相関は崩れている．あまり与えずに多く受取っているエージェントが増えている．科罰－不平等条件の図では，白い丸が強さ１のエージェントの度数を，黒い丸が強さ３のエージェントの度数を示す．科罰－平等条件と同様に，授受間の相関は崩れており，もともと強いエージェント（黒丸）が多くを受取り少ししか与えない側にいて，弱いエージェントがその逆であることが分かる．

３．４．授受関係の推移

資源のやりとりの不均衡がどのように生じるかを見るために次のような分析を行った．ラウンドブロックごとに，与える量と受取る量をそれぞれ多／中／少に３分類した．そして特定のラウンドブロックであるセルにいるエージェントが，ブロック変化時にどのセルから流入したかを調べた．結果の概要を図示したのが図９と図１０である．

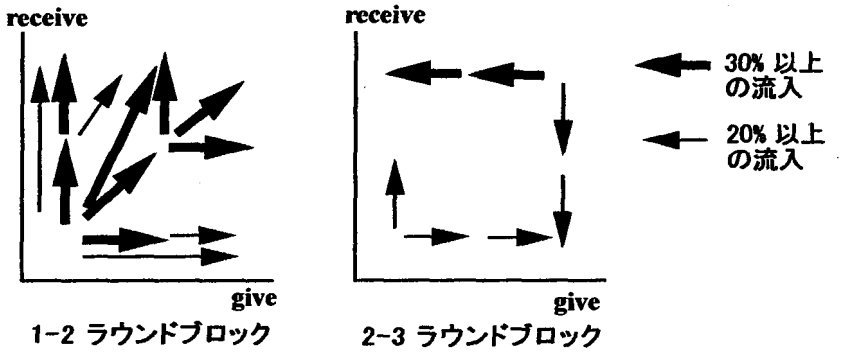


図9:エージェントの流入
（科罰－平等条件）

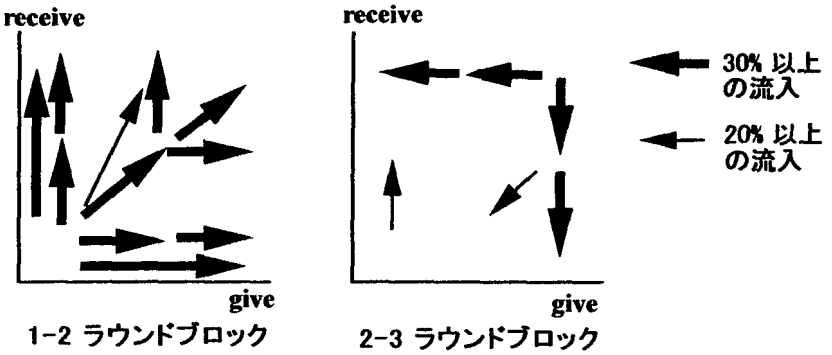


図10:エージェントの流入
（科罰－不平等条件）

科罰－平等条件の場合（図９），第１ラウンドブロックから第２ブロックにかけて，エー

ジェントは与える量と受取る量の両方もしくは一方が増大する方向に移動している．したがってこの時点では与える量と受取る量の相関はまだ高い（図７）．しかし第３ブロックになるとエージェントの流れが変わる．受取る量と与える量をいったん高めたエージェントのうち，「成功者」は与える量を減らし，「失敗者」は受取る量を失うことになる．その結果受取る量と与える量の相関は大幅に低下する（図７）．第３ブロック以降もおおまかには，以上の傾向が継続する．

結果は科罰－不平等条件でも本質的には変わらない（図１０）．

３．５．科罰

科罰のある両条件において，シミュレーションの設定通り，受取る資源が多いほど罰を受けることは少ない．最終ブロックでいえば，受取る量と被科罰度数の相関は，平等条件と不平等条件でそれぞれ，-.446 と -.514 (ps<.001)である．受取る量が多いほど科罰度数は高い(r=.239, .275, ps<.001)．

与える量と科罰との関連も弱いながら有意だった．与える量が多いほど科罰することは少ない(r=-.074, -.070, ps<.005)．また与える量が多いほど，平等条件では罰を受けることが少なく(r=-.093)，不平等条件では逆に多い(.113)．

なお不平等条件では，ラウンドを通して，もともと強いエージェント（強さ３）は科罰が多く被罰は少ない(χ^2検定)．

３．６．戦略スキーマ

１０の Run を条件ごとに合計し，エージェントの戦略を第２０，４０，６０，８０，１００ラウンドについて調べた．まずどのような戦略のスキーマが取り出せるかを分析した．戦略が４つの２進数からなる場合，例えばスキーマ '10**' とは，最初の２つが '10' であれば他の２つは任意で真となるパタンである．スキーマの非* 要素の数を仮にそのスキーマの次数と呼ぶ．全戦略の１／３以上が該当するスキーマを取り出す計算手順にしたがった．

３条件ともエージェントは同じ１６次元の戦略を持つ．しかし科罰なし条件では科罰戦略の６次元は無効であり，罰は生じないから交換戦略のうち罰を受けたか否かの次元も無効である．そこで有効な６次元（交換資源量の２次元＋強弱（２）×資源をくれたか否か（２））だけを取り上げる．見出せた最高次のスキーマは'1111*0'だった（35.6%，交換資源を最大に設定し，自分より強い相手には無条件に与え，弱くて自分にくれない相手には与えない）．予期に反して返報的なスキーマ(例：111010)は取り出せなかった．

この条件では科罰は生じないので強さは影響しないと当初予想した．が，強い相手には与える傾向が生じている．その理由は，「強さ」が相手の羽振りの良さ，つまり交換量の多さを意味する（授受の相関が高いので，多く受取るなら多く与えている）からだと解釈できる．第３－４次元の'11'は，交換資源を多く設定する戦略を持つエージェントに資源を

与える戦略であることを意味する．交換資源が多いエージェントが増えれば自分が受取る可能性も上がるだろう．また平均的エージェントにとって過半数の他者が該当する「弱者」に対しては，「くれなければ必ずあげない」という，部分的は返報性をとっていることになる．

科罰－平等条件では，最高次のスキーマは次の２つの６次のスキーマだった．'11*1*1*0*0******'(34.0%) と '**********000000'(33.6%)である（重複は 12.0%）．第１のスキーマは，交換資源を最大にし，自分に資源をくれるかどうかにかかわらず，自分に罰を与えない者の中で強い相手には与え，弱い相手には与えない強者優遇スキーマである．科罰は任意である．第２のスキーマは逆に交換について規定せず，誰にも罰を与えない無科罰スキーマである．調べた５ラウンドのデータを合わせて考えると，両スキーマの保有には相関がない．

各スキーマについて，そのスキーマに合う戦略の保有（２）×与える量（３）×受け取る量（３）に多重クロス表分析を適用した．両方のスキーマについて３重の交互作用が有意だった．与える量に比べて受け取る量が多いエージェントと，逆に過小受領するエージェントが，強者優遇スキーマを持つ傾向がある．ラウンドによっては罰を与えることが有意に多く，受けることは少ない．無科罰スキーマは与える量と受け取る量が均衡したエージェントを特徴づける．同時に，このスキーマのエージェントは罰を受けることも有意に少ない．

スキーマとしては析出できなかったものの，戦略の値の度数を見ると次の傾向が５つのすべてのラウンドで確認できる．強さが前試行で自分より高ければ協力するけれども，前回同等以下だった相手に対しては，罰を受ければ与える傾向があった．したがって罰はある程度，相手の協力をコントロールするように作用している．

総合すれば，この条件ではランダムな科罰がある状況下で強い者にすり寄る戦略（強者優遇戦略）と，科罰から逃れて平和に暮らす戦略（無科罰戦略）が析出されたことになる．

科罰－不平等条件でも同様の分析を行った．最高次の６次のスキーマは科罰－平等条件の無科罰スキーマだけだった(35.9%)．ただし４，５次のスキーマには強者優遇スキーマに近いスキーマが多くある．ここでは '11*1*1**********'（38.5%），つまり「交換資源を最大にし，自分に資源をくれるかどうかにかかわらず，自分に罰を与えない者の中で強い相手には与えるスキーマ」を強者優遇スキーマとして分析する．この科罰－不平等条件では，両スキーマの保有は負の関連をしている．

スキーマに合う戦略の保有（２）×与える量（３）×受け取る量（３）の多重クロス表分析を適用すると，強者優遇スキーマでは３重の効果が，無科罰スキーマでは２次までの効果が有意だった．強者優遇を特徴づけるのは受け取る量が中間的で与える量が多いエージェントである．強さがもともと低い（１）エージェントが強者優遇になることが有意に多い．無科罰スキーマは与える量が少ないエージェントに多く，もともと強い（３）エージェントが保有しやすい．科罰しないと同時に罰を受けることも少ない．科罰－平等条件

と同様に，強さが前試行で自分より高ければ協力するけれども，同等以下だった相手に対しては，罰を受ければ与える傾向が確認できた．

科罰－不平等条件では，強さ３のエージェントは（無科罰スキーマの担い手を出す一方で）主たる権力者になったと推測できる．強さ３のエージェントは１のエージェントに比べ，後半のブロックでは有意かつ顕著に，少なく与え多く受取り，科罰が多く被罰は少ない．強者優遇スキーマを持つエージェントはこのもともとの強者の権力を支持する（貢ぎ物をする）立場だったろう．もともとの強者のいない科罰－平等条件では，その授受のパタンから考え，強者優遇戦略の担い手の中から中心的な権力者を出したと考えることができる．

４．考察

４．１．結果の概要とその含意

自分にとって望ましい行動を他者にとらせる可能性を権力を呼んでおこう．この研究のシミュレーションでは，他者による望ましい行動とは協力の供与に他ならない．このように見たとき，このシミュレーションが示したのは，科罰可能性を導入したときに権力が偏在するという傾向である．

科罰可能性がないとき（科罰なし条件）には授受の相互性が成り立った．多くを受取るためには多くを与えなければならなかった．全体として相互性が成り立つ社会では，偶々生じた過大（ないし過小）受領はほどなく解消されてしまうだろう．

だが科罰が可能な（そして実際にか罰が生じる）条件ではこうした相互性は全体として崩れてしまう．つまり与えずに多くを受取る者と，多くを与えながら受取ることが少ない者を生じさせる．しかも過大（過小）受領状態は特定ラウンドで偶々生じる訳ではなく，ある程度のラウンドで持続する．

科罰条件における相互性の欠如は戦略の面からある程度理解できる．おおまかには，前試行で強かった相手には与え，前試行で弱くても今回強く自分に罰を与える者には与える傾向が生じている．実際，過大受領したエージェントは罰を与える傾向があり，この傾向は不平等条件でもともと強いエージェントで高かった．つまりもともとの，ないし交換から生じた強さは交換の流れを，強いエージェントに有利な方向に導いている．重要なのは，元来の強さが平等の条件でも，交換の偏りを経て権力が生じ得たことである．

以上の結果は当初想定した権力の再帰的メカニズムが論理的に可能であることを示している．つまり，交換において他者の協力を得るのに成功することが権力の基盤となり，さらに他者の協力を得やすくする．

興味深いのは，第２ラウンドで与える量と受取る量が高かったエージェントの中から第

３ブロックでの成功者と失敗者が出ることである（図９，１０）．成功者とは，交換において相互性を実現し多くを受取る地位をいったん築いた後に，手のひらを返して与えなくなる（ことに成功する）エージェントである．この部類のエージェントは「権力の腐敗」（アクトン卿）を体現していると考えられる．

このシミュレーションには予想外の結果もあった．資源をくれない相手を罰し，くれる相手は罰さない，という制裁的科罰戦略がエージェント間に生じなかったことである．戦略の値の度数から判断すると，この制裁的科罰傾向は科罰－不平等条件でごく弱く生じていたに過ぎない．にもかかわらず「強い」エージェントに与えることがおこり得たのはなぜか？　１つの可能性は，潜在的な罰の被害によって強者優遇傾向が生じたことである．シミュレーションの科罰のルール（２．４．）では，弱者に協力しその弱者が罰を受けた場合，協力者も罰の被害を負担しなければならない．しかも弱者は協力者が少ないから，被害の分担率は高くなる（逆に強者への協力では科罰コストの分担率は低い）．簡単にいえば，下手に弱者に協力して被害を抱え込むよりは，強者に対してだけ協力した方がリスクが低くなる，という訳である．このメカニズムの帰結は弱者の孤立が補強されることである．

以上の解釈が妥当するなら，このシミュレーションで生じたのは，権力者が逆らう者を罰し迎合する者を許す，という情景ではない．多くを受取る権力者は確かに科罰することが多い．が，その科罰は弱い者を見つけては気まぐれに罰するのみである．そしてこの科罰の事実があることによって，エージェントは弱者に協力することを控え，結果として権力者への協力が確保されることになる．

４．２．今後への課題

本稿のミュレーションは，交換ネットワークの偏りから権力が生じるという経路が思考実験で可能であることをデモンストレイトしている．その意味で本稿のシミュレーションには何がしかの意義があっただろう．

が，満足できるものではない．今後改善すべき点を列挙してみよう．

第１に，本稿のシミュレーションではエージェント間に安定的な，閉じた交換関係（限定交換）が発達しなかった．今から考えれば，このシミュレーションで用いた交換戦略は，通常出現が想定される安定的な限定交換を発生させる条件（高木，1994）を欠いている．したがって権力のコアとなるような集団の形成は不明確だった．交換戦略の設定を改善することで，より明確な権力の発生を示せるかも知れない．

第２に，交換と科罰の関係の設定にも再考の余地がある．例えば，報告したシミュレーションでは，資源を労力と考えながら，資源は受取るだけで利得があると考え，その利得は科罰を実行しても減ることはない，という前提を用いている．しかし科罰をすれば労力を使うので，受取った資源の科罰参加分は利得から外すべきだったかも知れない．

第３に，進化のルールにも再考の余地があるだろう．本稿のシミュレーションモデルでは，利得が低かったエージェントは単に高利得エージェントの戦略を採用する，という仮

定しただけだった．しかしエージェントの置かれた状況を考えずに，単に他の好成績者の戦略を真似ることは無意味かも知れない．戦略が有効性は一般に，接触する他者の戦略分布に依存するからである．したがってエージェントの個別の状況に応じた「戦略学習」が生じるような前提をモデルに組み込む必要がありそうである．

第 4 に，空間特性にも工夫が必要だった．本稿のシミュレーションではエージェントはローカルな範囲でしか行動しない．だから覇権を握った後の劉邦のような巨大な権力者は出現できない．また，エージェントが移動するという前提も置くべきだったかも知れない．

注

(1) この研究は次の科学研究費によって補助を受けた．基盤研究 (C)，12834004.

(2) 権力の発生を扱うシミュレーションモデルはいくつか存在している．私が調べた限りで，まじ Doran ら (1994, 1995, 1996) は実在した過去の具体的社会を想定して，リーダーのいる協力集団の出現を DAI モデルで導いている．ただしモデルの詳細が記載されていないため，この研究の評価は私にはまだできない．また， Axelrod (1995, 1997) は次のような権力構造のシミュレーションを行っている．輪状の一次元空間に位置するエージェント (国家など) が近隣エージェントに貢ぎ物を要求できる．要求を受けたエージェントはしたがうか戦うかを決定し，負ければ貢ぎ物を出す．貢ぎ物のやりとりがあるエージェント間では戦いでも同盟する．このシミュレーションは多様なシナリオを産出している．１つの覇権国が成長することもあれば「帝国の過大拡張」によって最大の覇権国が凋落することもある．この研究は魅力的であるけれど，「負ければ貢ぎ物を出す」ことがモデルの中で前提となっているため，このシミュレーションで権力が創発的に発生したとはいえない．

参考文献

Axelrod, R. (1995) A model of the emergence of new political actors. In G.N. Gilbert & R. Conte (Eds.) *Artificial Societies.* London: UCL Press, Pp.19-39.

Axelrod, R. (1997) *The Complexity of Cooperation.* Princeton, NJ: Princeton Univ. Press.

Blau, P.M. (1964) *Exchange and power in social life.* Wiley. (ブラウ, P. 『交換と権力』, 間場寿一・居安正・塩原勉 (訳), 新曜社, 1974)

Doran, J. (1996) Simulating societies using distributed artificial intelligence. In K.G. Troitzsch, U. Mueller, G.N. Gilbert & J.E. Doran (Eds.) *Social Science Microsimulation.* Berlin: Springer, Pp.381-393.

Doran, J. & Palmer, M. (1995) The EOS project: Integrating two models of Palaeolithic social change. In G.N. Gilbert & R. Conte (Eds.) *Artificial Societies.* London:

UCL Press, Pp. 103-1325.

Doran, J., Palmer, M., Gilbert, N. & Mellars, P. (1994) The EOS project: Modelling Upper Palaeolithic social change. In G.N. Gilbert & J. Doran (Eds.) *Simulating Societies.* London: UCL Press, pp. 195-221.

Emerson, P.M. (1972a) Exchange theory, Part I: A psychological basis for social exchange. In J. Berger, M. Zelditch, Jr. & B. Anderson (Eds.) *Sociological theories in progress.* Houghton Mifflin, Pp. 38-57.

Emerson, P.M. (1972b) Exchange theory, Part II: Exchange relations and network structures. In J. Berger, M. Zelditch, Jr. & B. Anderson (Eds.) *Sociological theories in progress.* Houghton Mifflin, Pp. 58-87.

Epstein, J.M. & Axtell, R. (1996) *Growing Artificial Societies.* Washington, DC: Brookings Institution Press.

French, J.R.P., Jr. & Raven, B.H. (1959) The bases of social power. In D. Cartwright (Ed.) *Studies in Social Power.* Ann Arbor: Institute for Social Research, Univ. of Michigan, Pp. 150-167.

Haslam, N. (1997). Four grammars for primate social relations. In J.A. Simpson & D.T. Kenrick (Eds.) *Evolutionary Social Psychology.* Mahwah, NJ: Laurence Erlbaum Associates. Pp. 297-316 (Chap. 11).

Messick, D.M., Wilke, H., Brewer, M.B., Kramer, R.M., Zemke, P.E. & Lui, L. (1983) Individual adaptations and structural change as solutions to social dilemma. *Journal of Personality and Social Psychology, 44,* 294-309.

Ostrom, T.M. (1988) Computer simulation: The third symbol system. *Journal of Experimental Social Psychology,* 24, 381-392.

Sigrist, C. (1967) Regulierte anarchie. Walter-Verlag. (ジークリスト, C. 『支配の発生』, 大林太良・石川晃弘・長谷川博幸・岡千曲 (訳), 思索社, 1975)

Suleiman, R. & Fischer, I. (2000) When one decides for many: The effect of delegation methods on cooperation in simulated inter-group conflicts. IIPDM Report No. 162, Institute of Information Processing and Decision Making, University of Haifa, Israel.

Taber, C.S. & Timpone, R.J. (1996) *Computational Modeling.* Thousand Oaks: Sage.

高木英至 (1994) 社会的交換のシミュレーション・パラダイム. 埼玉大学紀要、30、23-55.

高木英至 (1999) 「コミュニケーションネットワークにおける創発的集団構造 ― シミュレーションによる分析」, 『社会心理学研究』、14巻、3号、113-122.

Thomas, L.C. (1984) *Games, Theory and Applications.* NY: Wiley.

Weber, M. (1922/1947) *Wirtschaft und Gesellschaft.* Abteilung, J.C.B. Mohr, T*bingen, 3. Aufl. (ウェーバー, M. 『権力と支配』, 濱島朗 (訳), 有斐閣, 1967)

I . 2 . A computational model of the emergence of social power

Eiji TAKAGI (Saitama Univ., Japan)

Abstract. This study examines the idea that social power emerges from a network of social exchange. Here, strong power is considered to be the agent's ability to gain others' cooperation without giving others its own cooperation. Suppose a situation where a set of agents are engaged in social exchange of cooperation among them. The agents, who are successful in gaining much cooperation from others through exchange, can manipulate strong coalitions and become powerful because they can punish others more effectively. If they have attained a reputatoin as strong, they can get cooperation from weaker others without giving their cooperation to others including their ex-exchange-partners. It is because weaker agents cannot resist strong ones, and because, if weaker agents give cooperation to weak agents, they may be punished by strong agents. Computer-aided thought experiments demonstrated that such notion worked fairly well.

One of the universal social facts is that some are more powerful than others. This study attempts to explain why such social power comes to exist. The classical power-dependence paradigm might argue that power comes to exist because subordinates are dependent on the powerful. What remains to be answered is such a question as why the structures where some are dependent on the others are constantly reproduced.

My idea goes as follows.

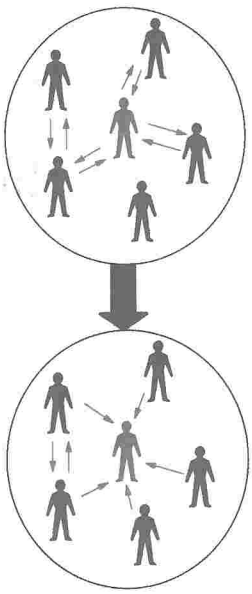
Assumptions:

(1) An agent can decide whether or not to cooperate with another (social exchange). It is a PD-like situation.

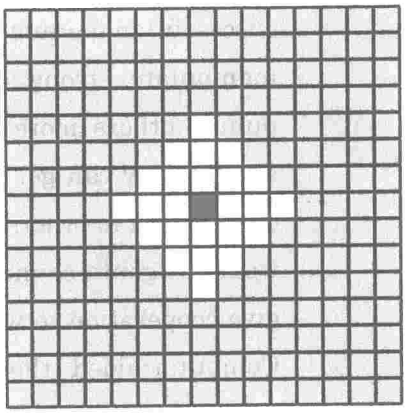
(2) An agent gains its strength if it can obtain cooperation from others.

Then, an agent who is successful in getting cooperation from others through social exchange can become powerful. Once it has attained a reputation as powerful, it can entertain cooperation from the weaker without being cooperative with them. Here, social power is conceived as an ability to get others' much cooperation in exchange for its own little cooperation.

This study examines if the above idea works well through a computer simulation equipped with evolution mechanism.



A Simulation Model



A torus 2-dimensional space: 15 x 15 cells (agents).

Rules for agents

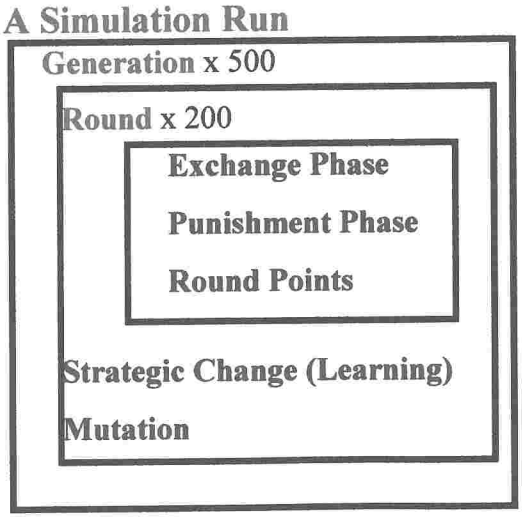
(1) An agent can interact only with others within 3 block-distance.

(2) An agent's strategy has two components

(a) Exchange component designates to whom its carrier agent give cooperation (labor).

(b) Punishment component dictates whom it punishes.

Rules for simulation (⇒)



Factorial design

Punishment factor: Punishment / No-punishment treatment

Strategy factor: Simple / Resource-conditional treatment

Strength of an agent

(1) Each agent is assigned an initial strength value.

(2) If agent A cooperate with agent B, B gains the half of A's initial strength.

Punishment rules

(1) A can punish B only if A is stronger than B.

(2) If A has cooperated with B, A does not punish B.

(3) The cost of being punished is much larger than the cost of punishment. The half of the cost is shared by the cooperators of the punishing/punished agent.

Description of a strategy

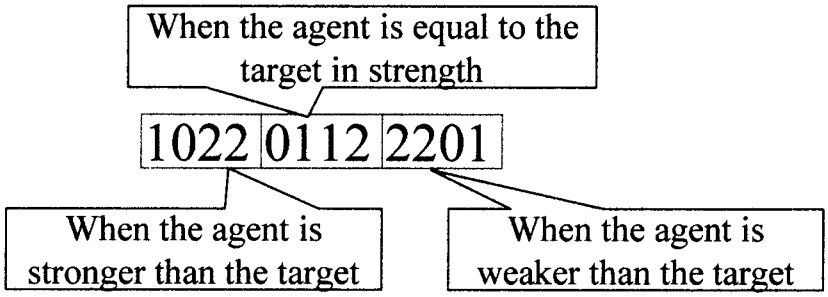
Depending on the value of the 'switch' dimension of a strategy, an agent's strategy can be a 'simple strategy' or a 'complex' one.

■Simple Strategy

●Exchange component

0: never give **1: give unconditionally**

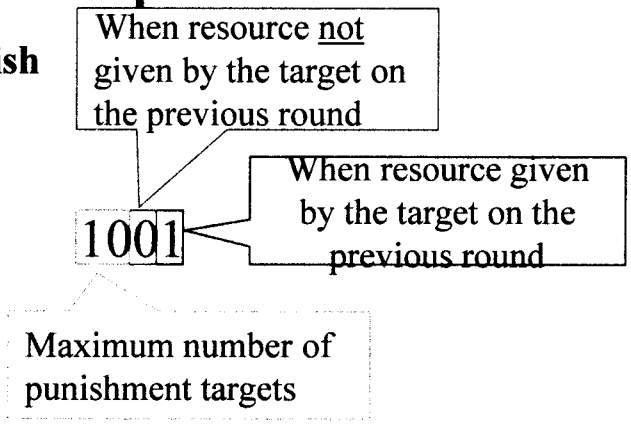
2: give if the target choose 1 or 2



●Punishment Component

0: never punish

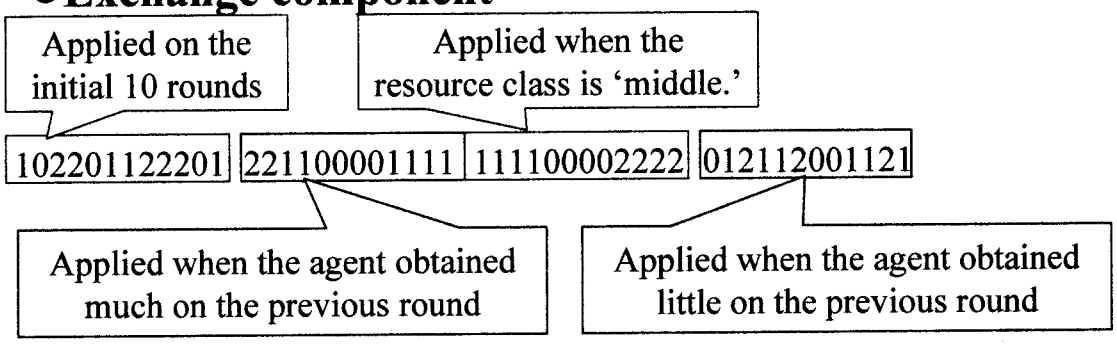
1:punish



■Resource-Conditional Strategy

●Applicable strategy depends on the total amount of resource the agent obtained from others on the previous round.

●Exchange component

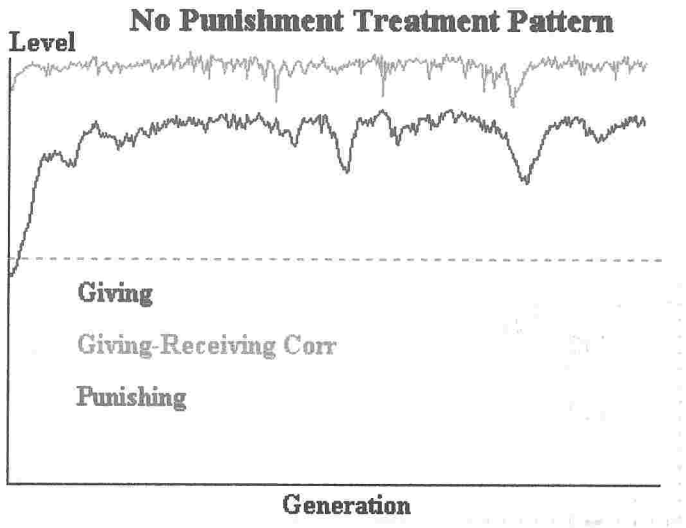


●Punishment Component

0101 1111 0000 1010

Results

In the **No-Punishment condition**, reciprocal exchange prevails. A high level of giving is maintained throughout a run. A agent's giving is well reciprocated (a high correlation between the giving level of an agent and its receiving level).

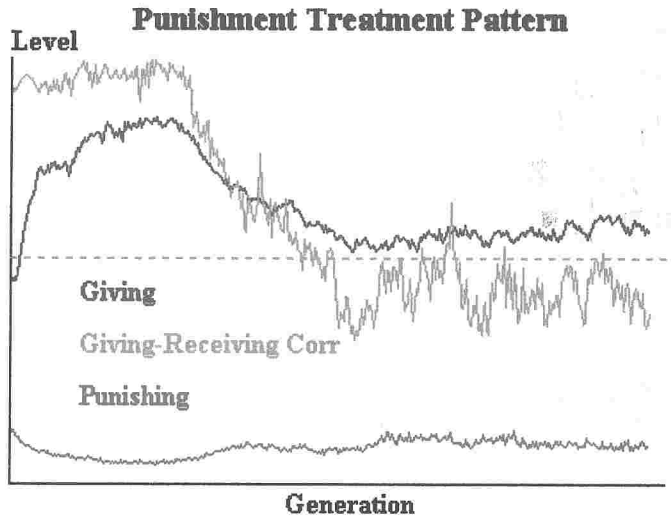


With Punishment:

(1) The level of giving decreases.

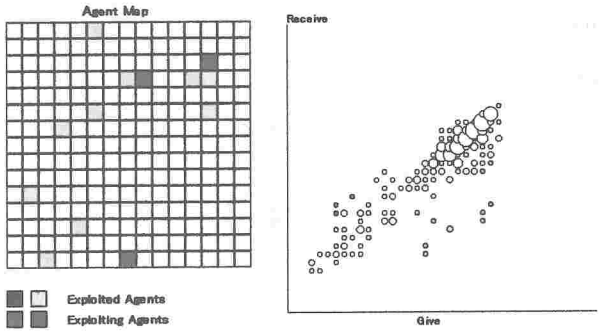
(2) The correlation between the giving level and the receiving level tends to be negative.

(3) There appear the exploiting agents (who give little but obtain much), and the exploited.

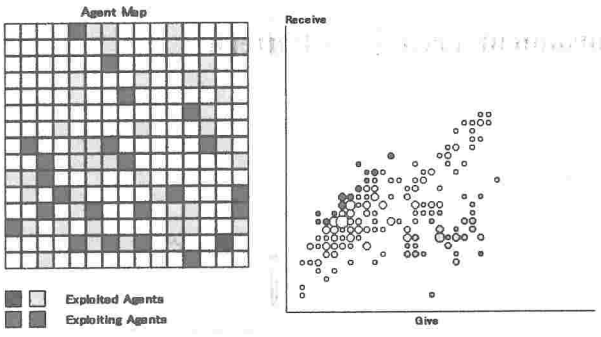


Differentiation of agents during a generation (an example).

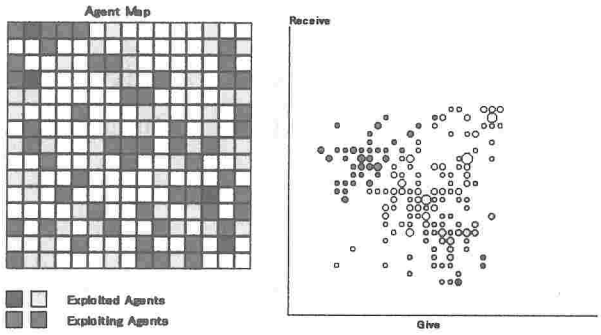
Under the Threat condition, differentiation of agents during a generation occurred as the following figures show. The blue (cyan) agents are exploiting (slightly exploiting) agents, who receive much resources while giving a little. The red (fuchsia) agents are exploited (slightly exploited) agents who give much resources but receives a little. Such differentiation is caused by the fixed sytaregies which the same agents carried.



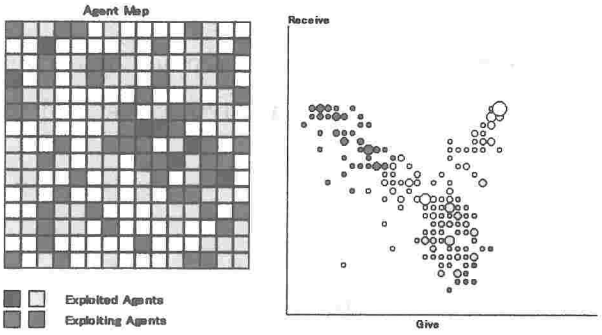
1st round



2nd round



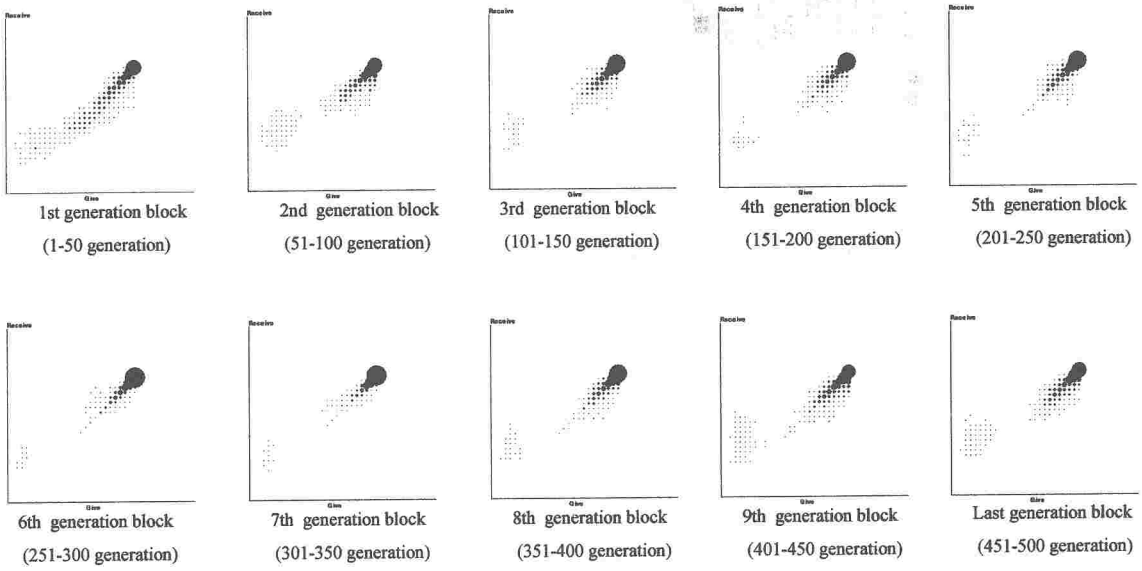
3rd round



200th round

The evolution of strategies across generations in a simulation run (an example)

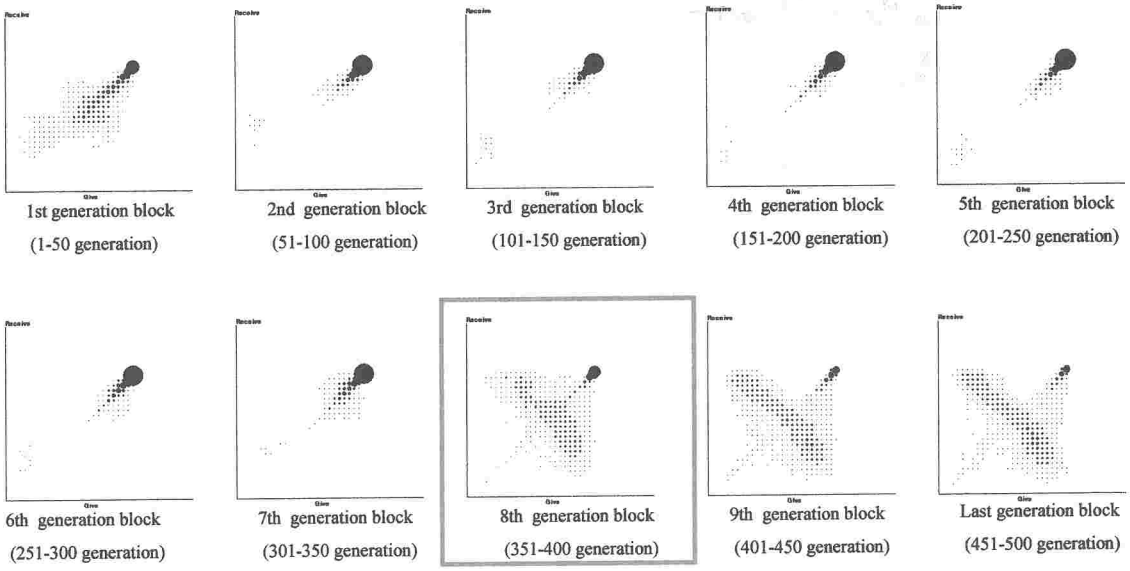
In the **No-Punishment condition**, an agent who gives much cooperation to others is likely to receive much cooperation throughout a simulation run.



1st generation block (1-50 generation) | 2nd generation block (51-100 generation) | 3rd generation block (101-150 generation) | 4th generation block (151-200 generation) | 5th generation block (201-250 generation)

6th generation block (251-300 generation) | 7th generation block (301-350 generation) | 8th generation block (351-400 generation) | 9th generation block (401-450 generation) | Last generation block (451-500 generation)

In the **Punishment condition**, a change took place at certain point (the 8th generation block in this example). Thereafter, the giving level and the receiving level tend to be negatively correlated.



1st generation block (1-50 generation) | 2nd generation block (51-100 generation) | 3rd generation block (101-150 generation) | 4th generation block (151-200 generation) | 5th generation block (201-250 generation)

6th generation block (251-300 generation) | 7th generation block (301-350 generation) | 8th generation block (351-400 generation) | 9th generation block (401-450 generation) | Last generation block (451-500 generation)

The evolution of strategies

The strategy patterns which evolved are rather complex. Broadly speaking, the following tendencies emerged.

(1) Threat-type punishing strategy:

'If you don't give me, I will punish you.'

(2) Slimy giving strategy:

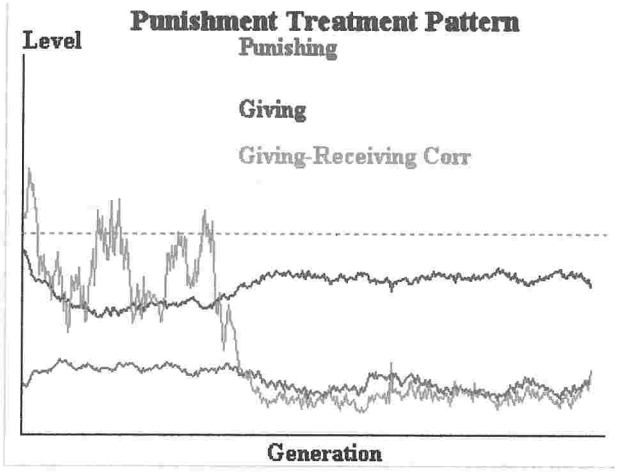
'I will give nothing to weaker others. I will give unconditionally to stronger others holding a threat-type punishing strategy.

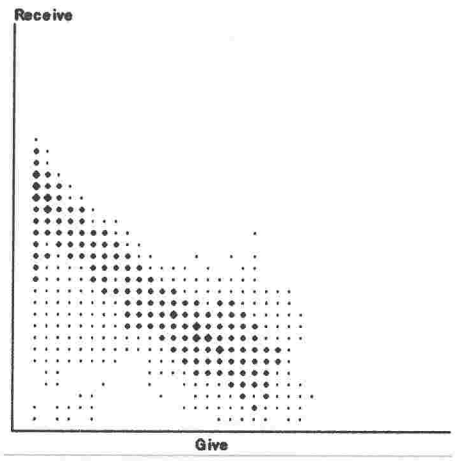
Discussion

(1) By simply introducing the possibility of punishment, power difference among the agents was observed.

(2) Strong reciprocity orientation might inhibit the emergence of power.

Reciprocal orientation in agents strategies tends to prevent a negative correlation between the giving level and the receiving level. To demonstrate this, I defined the values of exchange strategy only as 0 or 1 in order to remove the reciprocity-seeking tendency on the side of the agents. Expectedly, the correlation became more negative, and the agents are more likely to be either the exploiting or the exploited.





Ⅰ．３．社会的交換に基づく権力の発生

高木英至(TAKAGI, Eiji)
(埼玉大学 教養学部)

キーワード：交換，権力，コンピュータシミュレーション

【要約】社会的交換状況で、相手に資源を与えることなく相手から資源を得る能力を権力（power）と考える。この研究は、交換状況に相手を罰する可能性を導入するだけで、初期状態で強さの等しいエージェント間に権力の差が生じ得ることを計算実験によって示す。

背景にある問題関心は社会の基本的秩序を成立させる論理を見出すことにある。ここに基本的秩序とは、一定の範囲のエージェント間に協力性・利他性が成り立つこと、規範が作用すること、などを指す。社会に権力が生じることもまた、こうした基本的な社会秩序の１つである。

社会秩序の成立を説明する明確な方法はエージェント間の戦略的相互作用を経た動的均衡としてその秩序が成り立ち得ることを示すことである。この研究で用いるコンピュータシミュレーションは動的均衡としてある秩序が成り立つか否かを論理的に確認するための思考実験である。

以下では次のような権力の成立経路を検討する。各エージェントは他者に協力（労働力の供与）をするか否かの意思決定に直面するとしよう。また、各エージェントは他者に罰を科することができると仮定しよう。さらに、他者の協力が得られれば、エージェントは強さ（科罰の基盤）を増大させると考えよう。このとき、社会的交換（協力の交換）によって他者から協力を得ることが出来たエージェントは強さを増し、科罰能力を高める。一度科罰能力を高めたエージェントは、その可罰能力によって、自ら与えることなく他者からの協力を得ることができるようになる。逆に初期に協力を得られず弱い立場になった者は、その協力者も一緒に強者から強い罰を受ける可能性があるため、ますます協力は得にくくなるだろう。こうした経路でエージェント間に権力差が生じるか否かを思考実験することが以下の課題である。

［同じ目的の計算実験は高木（1999、社会心理学会大会報告論文集）で報告している。しかし高木(1999)では次の２点が改善すべき点だった。第１は、エージェント間の交換が、当初想定した限定交換（reciprocal な２者間の交換）には、設定上、なりにくかったことである。第２は、当初予想した「脅し型」の科罰戦略（協力しなければ罰する）が析出されなかったことである。以下の分析は上記の問題に対処するために行なった第２の試みである。］

モデルの設定

空間　１５×１５の torus 状のセル空間を想定する。各セルがエージェントを表す。各エージェントはブロック距離３の範囲の２４の他エージェントだけと相互作用する。

戦略　エージェントの戦略は科罰戦略と交換戦略の組である。科罰戦略は前回相手が自分に協力したか否かを手がかりに科罰する相手を指定する。相手が協力しなかったときだけ科罰することも可能であり（脅し型）、協力に関わらず相手に科罰する（あるいはしない）こともあり得る。交換戦略は相手の科罰戦略、相手と自分との強さの差（強／同／弱）によって相手を類別し、カテゴリー別に０（無条件に非協力）、１（無条件に協力）、２（条件付協力＝相手が０か１なら協力）を割当てる。

交換　協力には程度はなく、協力か非協力だけが生じる。相手に協力を控えれば、エージェントの得点は１向上する。しかし相手から協力を得れば得点は２向上する。従ってエージェントペア間には協力するか否かのＰＤ状況が存在する。エージェントが利己的に行動するなら協力は生じない。

強さと科罰　各エージェントは初期状態で等しい「強さ」の値（２）を持つ。が、他者からの協力を得れば、協力者の強さの半分の値（１）だけ強さを増す。罰は弱い相手に対してだけ科すことができる。科罰のコストは１、罰を受けるコスト（被害）は

$$5 + |\text{両者の強さの差}|$$

である。もし協力者があるなら、科罰コストの半分は当のエージェント、あと半分はその協力者が平等に分担する。罰の被害の半分は罰を受けたエージェントが引き受け、残り半分はその協力者が等しく分担する。

シミュレーションの構成　シミュレーションの１回の Run は 500 世代からなり、１世代は２００ラウンドからなる。１世代の終わりでエージェントの戦略分布の変化（学習）と突然変異が生じる。戦略分布の変化は、世代での合計得点が下位５％であったエージェントの戦略が上位５％のエージェントの戦略（からランダムに選んだ１つ）と入れ替わることである。突然変異は各戦略次元で等確率で独立に生じる。Run の初期状態では戦略の値は無作為化する。

１ラウンドには交換局面と科罰局面がある。交換局面では交換戦略に従ってエージェント間で協力のやりとりが生じる。その際、戦略が参照するエージェントの強さは前ラウンドの強さである（最初のラウンドでは強さは平等）。交換局面での協力のやりとりに応じてそのラウンドでの各エージェントの強さが計算され、その強さが次の科罰局面で参照される。そのラウンドでの各エージェントの得点は、交換局面での得点と科罰局面での（負の）得点の合計である。

実験計画　シミュレーションは２（戦略要因）×２（科罰要因）の実験計画で実施した。条件ごとに１０の Run を繰り返した。

戦略要因　戦略要因は、エージェントの戦略が単純か資源依存的かの２水準を持つ。単純戦略条件とは、科罰・交換戦略が記述の通りの構成の場合であり、エージェントの科罰戦略が２値の４次元、交換戦略が３値の１２次元で成り立つ。資源依存戦略条件とは、単純な科罰・交換戦略の４倍の次元を持ち、最初の１０ラウンドで適用される戦略、エージェントの得た協力が上位１／４のときに適用される戦略、獲得した協力が中間的であるときに適用される戦略、獲得した協力が下位１／４のときに適用される戦略に条件分化している。資源依存的戦略は、戦略分布を変化させる際でも、獲得した協力の類別内で優秀だったエージェントの同類別部分の戦略を受け継ぐように計算する。

科罰要因　科罰要因は、科罰可能性を導入した場合（科罰条件）と導入しない場合（科罰無し条件）の２水準である。どちらの水準でも、用いた戦略の構成は変わらない。科罰無し条件でもエージェントは科罰戦略を持ち、その交換戦略は相手の科罰戦略に応じて相手に協力するか否かを判断する。ただ、科罰無し条件では他者に実際に科罰することはない。

シミュレーション結果

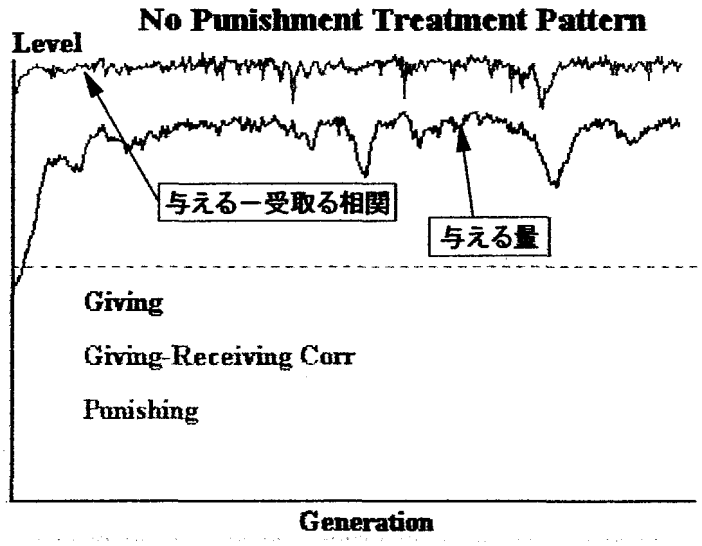


図１：科罰無し条件でのシミュレーション例

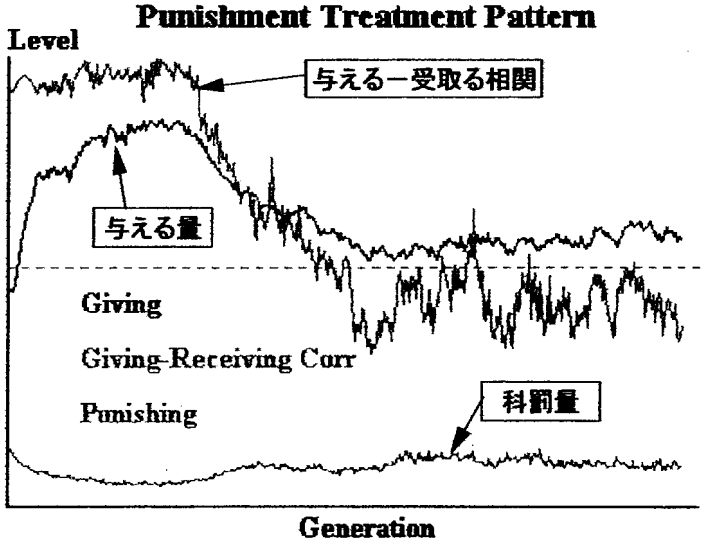
１０の世代ブロック（１ブロック＝５０世代）を繰り返し要因として、１エージェント当りの与えた協力量の（世代を通した）平均、エージェントごとの与えた量と受取った量の相関係数（ｚ変換後）を従属変数とした分散分析を実施した。戦略要因の主効果、戦略要因×科罰要因の交互作用効果は有意にはならなかった。効果を持ったのは科罰要因だけである。最終ブロックでの与えた協力量の平均は、科罰無し－単純戦略条件で 22.3、科罰無し－資源依存戦略条件で 22.1、科罰－単純戦略条件で 16.7、科罰－資源依存戦略条件で 15.2 だった。つまり科罰無し条件では与える量は低い。科罰条件での与える量の低下傾向は第６ブロックから生じている。エージェントごとの与えた量と受取った量の相関係数は

エージェント単位で与える量と受取る量のバランス（相互性）が成り立つ程度の指標となる。最終ブロックでのこの相関の平均値（z変換前で表記）は、科罰無しの２つの条件でともに .93、科罰－単純戦略条件で -.04、科罰－資源依存戦略条件で -.17 だった。つまり科罰無し条件では交換に高度の相互性が守られている。対して科罰条件では相関は負になっており、多くを受取りあまり与えない、あるいは多くを与えながらほとんど受取らないエージェントが出現していることを表している。なお、科罰量の同様の分析（科罰条件だけ）でも戦略要因は有意な効果を及ぼさなかった。

科罰無し条件での結果のパタンは図１で例示される。与える協力量は初期の世代で急速に上昇し、与える－受取る量の相関は終始 1.0 に近い数値を維持している。科罰条件での結果のパタンは図２が例示する。初期の世代では科罰無し条件と同様に、与える協力量は増大し、与える－受取る量の相関も高い水準を維持する。しかし何れかの時点で与える量の平均と相関は急激に低下し、相関は負になる。（科罰無し条件の２０の Run は何れも図１のパタンを示した。科罰－資源依存戦略条件では１０の Run が図２のパタンを示した。科罰－単純戦略条件では８の Run が図２のパタン、２つの Run で図１のパタンとなった。）

エージェントの最終世代での戦略パタンを分析した。次元の多い戦略であるため、戦略パタンはやはり複雑だった。大まかには次の傾向を観察できた。第１に、科罰条件では脅し型の科罰戦略（「私に協力すれば罰を与えない、協力しなければ罰を与える」）が最も優勢であり（図２のパタンを示した科罰－単純戦略条件の観測で５０．４%）、その次に優勢だったのは無科罰戦略（同、３６．０%）だった。他方、交換戦略では「俗物戦略（slimy giving strategies）」、つまり相手が自分より弱ければ全く与えず、強ければ無条件に与える戦略が優勢だった。

図２：科罰条件でのシミュレーション例



最終世代では次のようにして「権力の分化」が生じている。まず第１ラウンドではエー

ジェントの与える量と受取る量はほぼバランスしている。しかし第２ラウンドで与える／受取る量が少ないエージェントが増え、以後、受取る量が多く（少なく）与える量が少ない（多い）エージェントが増えてゆく。初期のラウンドでの授受バランスのわずかな偏りが以後の強弱の大きな差を生み出す格好になる。ただし一定数のエージェントは多く与え多く受取る状態を維持している。

考察

この計算実験の直接的な含意は、協力の交換が生じる状況で科罰の可能性を導入すると、ただそれだけで、一方では他者に与えず受取るだけのエージェントが、他方では他者からほとんど受取らずに専ら貢ぐエージェントが出現することである。社会的交換状況で与えずに受取ることが権力の現れであると解するなら、このシミュレーションが観測したのは局所的な権力の発生に他ならない。社会的交換は人間社会に普遍的な現象である。狩猟採集社会の時代から人間は社会的交換によってその存続を果たして来ただろう。ここで個人間のネットワーク（同盟関係）の偏りによって強さ（科罰能力）に差が出ると想定するなら、人間社会において権力が発生することは同様に自然な推論である。

用いたモデルはエージェント間で相互的な限定交換が生じやすく設定してある。そのため、科罰条件下でも権力分化状態以外に、科罰せずに相互的な交換が優越する状態も別の均衡点だったと言える。つまり相互的交換への志向性の強さは権力の発生を抑制するよう作用する可能性がある。実際、交換戦略で２（条件付協力）の値を排除し、交換の相互性を低下させたシミュレーションでは、科罰条件で与える量と受取る量の相関係数は -1.0 に近づき、権力の分化が顕著になる。この点は、限定交換が生じるときより一般交換が生じる社会圏で権力差が生じやすいことを意味しているかも知れない。

Ⅱ．１．Social Impact シミュレーションのタネと仕掛け

高木英至(TAKAGI, Eiji)
(埼玉大学 教養学部)

キーワード：Social Impact、シミュレーション

[要約] Nowak、Latanéらの Social Impact シミュレーションの結果は影響力の定式化やパラメータの値によって変異する。一般に理解されている結果は必ずしも頑健な傾向ではない。

Nowak、Latané らの Social Impact シミュレーション（以下ＳＩＳと略）の結果の大筋は次のごとくであると一般には理解されている。距離が定義される平面的空間に位置するエージェント（セル）がランダムに２値的な態度の何れかを持つ初期状態から出発して、（ａ）同じ態度を持つエージェントのクラスタが生じる、（ｂ）初期多数派はより多数に、初期少数派はより少数になる、（ｃ）[空間に端がない（torus でない）ことを前提に]少数派は端に位置しやすい、ことである。

しかしＳＩＳのプログラムの再現を試みた経験のある者は、このシミュレーションモデルがときおり不可解な挙動を示すことに気がつくと思う。最も目立つのは、ある条件下では少数派が初期状態より増えてしまうことである。

ＳＩＳモデルの挙動はむろんパラメータの値によって変わる。そこで以下ではパラメータの値を体系的に変えることにより、ＳＩＳの結果がどの程度変化するか、あるいはどの程度頑健かを検討してみる。

標準モデルによる分析

研究目的はＳＩＳの主要な知見の頑健性を評価することにある。しかしモデルの前提は初期の Nowak, Szamrei & Latané (1990)と Latané, Nowak & Liu (1994) でも異なる。そこで別に次の前提でモデルを組んだ。 50×50 のセル空間、ユークリッド距離、上下左右の隣のセルとの距離は１、セルの自分との距離は Latané, Nowak & Liu (1994) に従って 0.84 と仮定した。また各セルの「説得力」と「サポート力」は同一の「強さ」で表し、全セルの強さは同一(=1)とした。導入した要因は以下の２×２×２×４である：影響力モデ

ル（Faction-size／Accumulative influence）、空間形状（端がある／ torus）、態度更新方法（同時／ランダム）、初期少数派比率（１０%／２０%／３０%／４０%）。各条件で２０の run を試行した。全セルが変化しないラウンドが２回続いたときに run を打ち切った。

最終残存少数派数の分散分析結果は、２つの交互作用を除いてみな有意になる。大まかにいえば、過去の結果から予測できるように、Faction モデルのとき、端があるとき、初期少数派比率が高いときに、少数派は残りやすい。また全般的には、更新方法がランダムのときに同時より、少数派は残りやすかった。

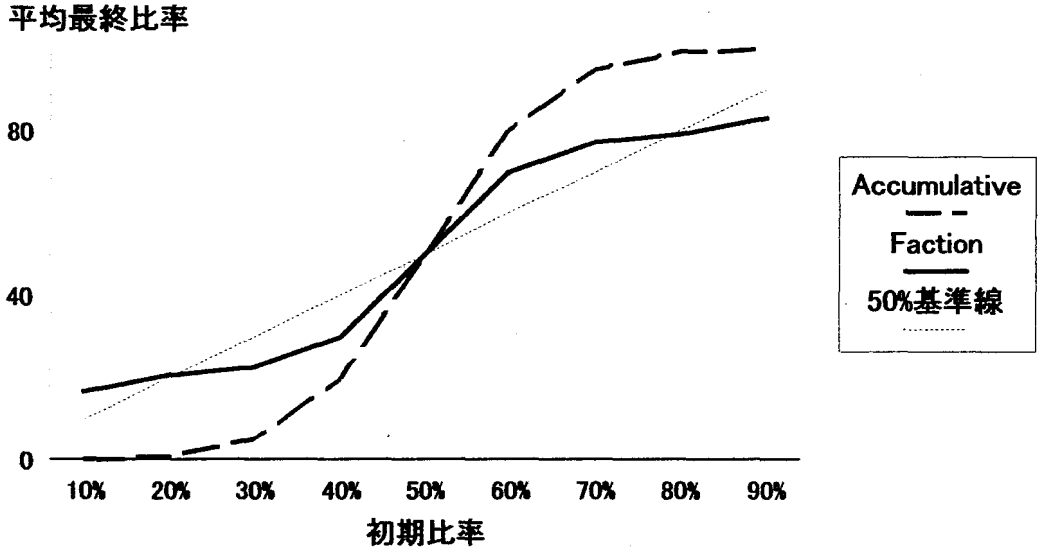


図１：初期比率と最終比率の関係（全モデル合計）

図１は空間形状×態度更新方法を込みにして影響モデル別に初期比率と最終比率の関係を描いたグラフである。Faction モデルがAccumulative モデルより少数派を残しやすいことを表している。注意すべきは、Faction モデルでは初期少数派比率が１０%および２０%のときに、少数派サイズが初期より最終で増えていることである。実はこの傾向は３０%だけの結果を示した Latané, Nowak & Liu (1994) のモデルを再現しても得られる。

距離係数の効果

影響力の定義式はFaction モデルでは(1)、Accumulative モデルでは(2)である。

$$i = N_0^{1/2} \cdot \Sigma(s_i / d_i^2) / N_0 \qquad (1)$$

$$i = [\Sigma(s_i / d_i^2)^2]^{1/2} \qquad (2)$$

ただし、s_i:セルの強さ、d_i:セル間の距離、N_0:特定の態度のセル数。自己の態度への i を他の態度の i が超えるときに当セルの態度が変化することは同じである。

(2)の右辺を1／2乗していることには実は意味がない。更新は i の大小だけで決まり、大小関係は1／2乗してもしなくても同じだからである。また、上の標準モデルのように強さ(s_i)を一定とすれば、(1)、(2)は次の式で考えても同じである。

$$i = N_0^{1/2} \cdot \Sigma(S_i / d_i^n) / N_0 \qquad (1')$$

$$i = \Sigma(S_i / d_i^m) \qquad (2')$$

上式のnとmを「距離係数」と呼び、その値を要因(5水準:0～4)として、ランダム更新、torus、少数派初期比率30、40%で同様のシミュレーションを行った。距離係数が0のとき、任意のセルに対する影響は全セルで等しい。距離係数の上昇とともに遠いセルからの影響力は低下する。距離係数が大なら、当然、多数派の影響力が弱いから少数派は残りやすい。隣接セルだけから影響を受けると仮定したとき少数派が残りやすいことを示した Latané らの結果も同じ理屈で理解できる。

Faction モデルではn＝0、1のとき少数派は全滅する。またn＝3、4では、再び、少数派の最終比率は初期比率を超えてしまう。つまりSISの基本知見が再現されるのは(1')でn＝2のときだけである。

Accumulation モデルではm＝0～3で少数派は全滅する(わずかに40%で出発したときに約0．5%が残る)。つまり(2')でm＝4としたのは少数派が残り得るようなパラメータ設定だったといえる。両モデルとも、距離係数を小さくすれば少数派クラスタは存在できない。

自分との距離を要因にして(.64～.94)同様のシミュレーションを行なった。自分との距離が小さければ当セルの態度は変化しにくい、という当り前の結果が生じた。Faction モデルではパタンは複雑であったけれど論ずるには足らない。

両モデルの問題点

run の過程は両モデルで大きく異なる。Accumulation モデルではラウンド進行とともに少数派が単調に減少し均衡に至る。しかし Faction モデルではほとんどのケースで第1ラウンド終了時に少数派サイズは最小になり(サイズが最小となる平均ラウンド数は 1.28)、以後少数派は盛り返す。極端な場合、第

図2：少数派の盛り返し
(Faction モデル、少数派の初期比率＝10%)

1ラウンド終了時に少数派が1セルだけ残ったときでも、その1セルが仲間を増やして行く。そして初期比率が10%、20%のときには最終比率が初期比率を超える(図2)。

少数派による奇妙なこの盛り返し現象が生じる理由は Faction モデルの定式の中にある。(1)は次のように書き換えることができる。

$$i = \Sigma(s_i / d_i^2) / N_0^{1/2} \qquad (1'')$$

この i が多数派の影響を指すとしよう。このとき、多数派は分子で加算される(s_i / d_i^2)の項が多くなるものの、距離の二乗で割っているので遠くの多数派セルからの影響は小さい。しかしその小さい影響しか及ぼさない多数派セルの数は分母の$N_0^{1/2}$でカウントされてしまう。従ってこの式は、多数派の影響を抑制するように出来ている。逆に少数派の影響では、分子が小さい(人数は少ない)ためにその影響力は大きく評価されることになる。

少数派の盛り返し現象の原因が以上の理由にあるとして、ではなぜ Faction モデルに従ったはずの Nowak, Szamrei & Latané (1990) において、少数派は常に初期より最終で小さくなったのか。筆者の検討では理由は2つある。第1は Nowak, Szamrei & Latané (1990) では特殊な距離の定義をしており、その定義が上記の距離係数を低くする(つまり少数派を抑える)のと同様に作用したことである。第2に同論文では影響源として計算するセルの範囲を距離10に限定したため、多数派の影響でも(1'')の分母が大きくならなかったことである。実際この2点を変更したとき、シミュレーション結果は図1の Faction モデルの場合と同じく、初期比率10、20%で少数派は平均して初期比率より多くなる。

このように考えると、Faction モデルはSISの基本知見(b)を自然に充たすとは考え難い。均衡に至るまでの過程(少数派の盛り返し)も一般的である証拠はない。

他方、Accumulative モデルの欠点は基本知見の(a)を明確に示さない点である。 Latané, Nowak & Liu (1994) の図で示されるように、このモデルでは明確なクラスタ化が生じ難い。基本モデルの結果を使い、最終的に少数派であったセルのうち最初から少数派であったものの比率を計算した。Accumulative モデルではその比率は 91.8% であり、Faction モデルでは 29.7% だった。つまり Faction モデルでは少数派も多数派を多く改宗させている。しかし Accumulative モデルでは多数派からの改宗者は極端に低い。この点は、Accumulative モデルでのクラスタ化が、多くの場合、孤立した少数派が抜け落ちたことの結果であることを表している。少数派固有のクラスタ化作用はほとんど働いていない。

考察

SISは単純な前提から重要な含意を提供している。エージェントの移動がないにもかかわらず意見によるクラスタが生じること(a)、および多数派の強い影響力と少数派の残存可能性とを「初期少数派の減少」(b)と表現したことである。

この報告の分析が示したのはSISモデルの挙動が距離係数、影響力の定式、空間形状、態度更新方式などの変更によって有意な変化が生じることだった。特にSISの基本知見が成り立つためには距離係

数(など)によって遠方のセルの影響が適度に遮断されることが必要だった。また、Faction-size モデルは顕著なクラスタ化を説明するには魅力的であるものの、その挙動は広く理解されているSISの基本知見とは食い違う面があった。

SISの含意を拡張する1つの方向は意見の数が2より大きい場合を考えることだろう。このとき、意見間のなんらかの距離を想定するか、意見が質的に異なる(従って距離は想定しない)かによってモデル構成は異なる。仮に後者の考え方に立ち、各セルにはそれぞれの意見に向けた影響力が働くと考えてみよう。

意見数を3とし、意見の初期分布を1:2:2としたときのSIS(Accumulative モデル)の結果(意見の最終分布)を**図3**に例示する。このとき最小集団は初期比率20%から出発し、最終では5. 44%に落ちる。しかし意見数2のときの対応する少数派最終比率0. 64%よりは有意に多い(*F(1,39)=241.14, p= .000*)。初期20%集団のうち最終分布で当初から同意見だったセルの比率も、2つの意見の場合(98.4%)よりの有意に低下する(83.9%, *p=.000*)。

初期意見分布が1:2:2の場合とは、意見数2で少数派が20%から出発するときの多数派がちょうど2分裂した場合に相当する。従って少数派が残りやすいことは当然といえる。しかし図3のパタンは1つの興味ある傾向を示している。最小意見のクラスタはほとんど、2大意見のクラスタの双方に接して存在していることである(ただしこの傾向は1つの意見が大多数となるときには見られない)。端のある空間形状のときに少数派が末端に位置しやすかった[基本知見(c)]ように、少数派は多数意見の谷間にニッチを見つけるのかも知れない。

しかしこの報告の分析はSISの基本的知見の(a)と(b)がパラメータの微妙なバランスの下に成り立つことを示す。特に元来のSI理論との関連が強い Faction モデルは魅力的な予測は産出していない。

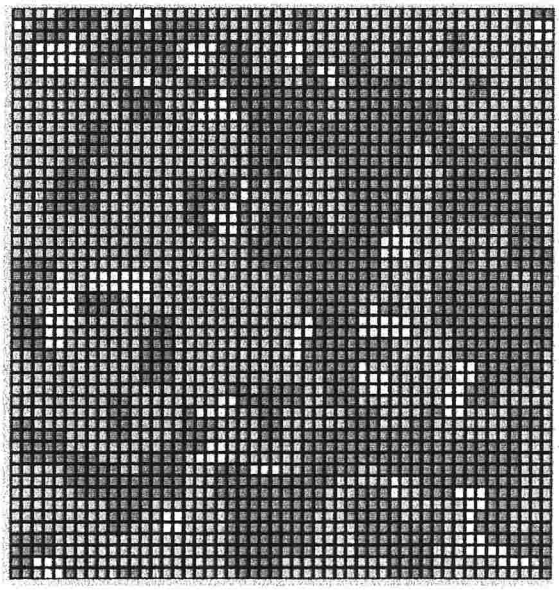


図3:3種類の意見の場合
(Accumulative モデル、初期比率は1:2:2)

Ⅱ．２．相互調整によるエージェントのクラスタ化：コンピュータシミュレーションによる検討

高木英至（TAKAGI, Eiji）
（埼玉大学　教養学部）

キーワード：相互調整，社会的影響，コンピュータシミュレーション

意見や価値観などの態度が類似した者がその態度に応じてクラスタ化（clustering），あるいは分離（segregation）を達成することは，社会の基本的秩序の１つである．こうしたクラスタ化は大別して２つの経路から生じ得る．第１は移動によって態度が類似したエージェント同士が近接して位置するようになることである．Schelling (1971)、Sakoda (1971) の説に従ったコンピュータシミュレーション（Epstein & Axtell, 1996）は，自分と同じ値を持つ隣接エージェントの比率への選好を個々のエージェントが持つとき，エージェントの移動によってエージェント社会に分離ないしクラスタ化が生じることを明らかにしている．第２は相互の影響によってエージェントの態度変化がおこることでクラスタ化が生じる経路である．Nowak, Latané らの社会的影響(Social Impact)のシミュレーションモデル（以下，ＳＩＭと略）は，セル空間上のエージェント間に距離に応じた影響力が生じることを前提に次の傾向が生じることを示した(Latané, Nowak & Liu, 1994; Nowak & Latané, 1994; Nowak, Szamrei & Latané, 1990; ）．（１）態度ごとのクラスタが生じる．（２）初期状態での少数派もクラスタになることによって残存できる．しかし均衡時の少数派サイズは初期比率に比べて減少する．（３）空間に端がある（空間が torus でない）とき，少数派は空間の端に位置しやすい．

ＳＩＭは態度が２値的な場合に限定されるなど，強い制約の下にある．そこで以下ではＳＩＭをより一般化した「相互調整モデル(Mutual Adjustment Model, ＭＡＭと略)」を提示する．ＳＩＭとの大きな相違は次の２点である．第１に，ＳＩＭでは態度が２値（賛成／反対など）であったのに対し，相互調整モデルでは態度を２進数の組（例：1001）と定義する．従って態度の値は多数になり得るし，態度間の距離も定義することができる．第２に，ＳＩＭでは影響力が強い態度をエージェントがそのまま採用すると仮定するのに対し，相互調整モデルではエージェントが態度の変更を自発的に試み，現在よりも適合度の高い態度に変化する，と考える．例えば，ＳＩＭでは２つの強力な態度勢力に出会ったときにエージェントは常にその片方を採用することになる．しかしＭＡＭでは，00 と 11 という勢力が強い態度の間に位置するエージェントは 01 といった折衷的な態度をとることが許される．

ＳＩＭとMAMの挙動の間には連続面と非連続面があると想定できる．連続面とは次の２点である．

第１に，少数派を含めて複数の態度のクラスタができるという結果は，影響力が局所的に限定される場合だけのはずである．セル間の影響が距離にかかわらず大域的に働くなら，あるいは距離による限定が少ないなら，セル空間内では最強の態度が全体を支配するはずである．ＳＩＭについては，「距離係数」（後述）が小さければ（大域的な影響が生じれば）少数派の残存可能性がなくなることが分かっている（高木，2000）．同様の効果はMAMでも観測できると予想できる．

第２に，初期の少数派が均衡時により少数になるという傾向もＳＩＭと同様にMAMでも観測できるだろう．この傾向は規模ないし度数に基づく構造効果と見ることができる(Blau, 1977; Mayhew & Levinger, 1976)．エージェントが態度にかかわらずランダムに分布するとき，多数派エージェントは周囲に多くの同類を見出し，少数派エージェントは多数派に囲まれやすい．従って改宗によって多数派は増大し少数派が減少する．このような多数派［少数派］の増加［減少］は乗数効果のように累積し，少数派が相互支持的なクラスタに縮小するまで続くだろう．同じ過程はMAMにおいても生じるはずである．

他方でMAMにはＳＩＭにない要素が入っている．最も重要な要素は態度間の距離である．態度は単に２値的に異なるのではなく、その間に何らかの距離を想定することは自然な仮定のはずである．かけ離れた態度はその保持者相互に適合度（後述）の低下をもたらす．従ってかけ離れた態度の一方が優勢になる状況ではもう片方の態度が特に劣勢になることが生じるかも知れない．また，かけ離れた態度がともに優勢なクラスタを作るとき，両者は牽制し合って中間的な態度がニッチを見出して生き残りやすくなるかも知れない．ＳＩＭではまだ検討されていないこうした側面に検討を加えることが本研究の重要な目的である．

相互調整モデルの前提と手順

MAMは以下の前提と手順に従う．

１．50×50 のセル空間を仮定する．各セルが１人のエージェントを表すと考える．エージェントは移動しない．セル空間は torus であり，上端と下端，右端と左端はつながっている．

２．セルの座標を行と列の順番で示す．左上のセルは（１，１），右下のセルが（５０，５０）である．

３．セル間にブロック距離を定義する．座標 (x_1, y_1) と (x_2, y_2) の距離を次式で表す．

$$d[(x_1, y_1), (x_2, y_2)] = \min \mid x_1 - x_2 \mid$$

$$+ \min \mid y_1 - y_2 \mid \qquad (1)$$

「隣接するセル」とは距離が最小（1）の他のセル，つまり上下左右４つのセルである（ノイマン近傍）．

４．セル（エージェント）の態度の値を２進数の組（例：’1001’）で定義する．任意のセル間の適合度を，態度の要素の値が一致する数で定義する．’01’ と ‘10’ 間の適合度は０，’11’ と ‘10’ の適合度は１である．不一致の要素数が態度間の距離を指す．

５．セル i $(=(x_1, y_1))$の態度の適合度を次式で表す．

$$\mathrm{Fitness}(i) = \sum_{j \in A} f_{ij} \cdot s_j / d_{ij}^{n}. \qquad (2)$$

ただし，Aはセル i が「相互作用」するセルの集合，f_{ij} はセル間の適合度，s_j はセル j の強さ，d_{ij} はセル間の距離である．s_jは全セル同一（1.0）と考える．記定式はＳＩＭの accumulative influence model (Latané, Nowak & Liu, 1994) に対応している．

６．（２）式の n を距離係数と呼ぶ．セル間の距離と影響力の潜在的な大きさの関係を表したのが図１である．n = 0 のときは，任意のセルは距離にかかわらず他のセルから等しい影響を受ける．n が大きくなるほど遠いセルからの影響は小さくなる．

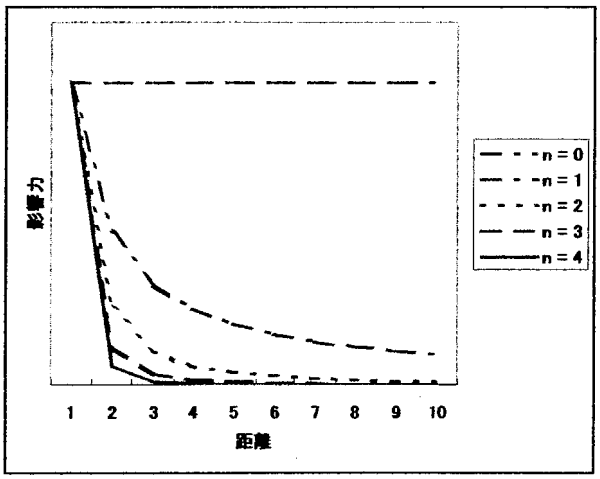


図１：距離係数の効果

７．現在の状態の「慣性」を考慮するため，ＳＩＭにならってＡには自分自身も含める．自分自身との距離には 1.0 以下の数値を当てはめる（後述）．

８．各セルは自分のターンが来たとき，態度の各次元を確率 0.5 で探索的に変化させてみる．この仮の態度値の適合度が現状より高いときだけ，その仮の態度値に変更する．

９．シミュレーションは離散的なラウンドによって進行する．セルのターンはラウンドごとに乱数で決める．全セルの初期の態度値も乱数で決める．態度値を変化させるセルが全く出現しないラウンドが3回続いたとき，均衡にあると考えてシミュレーションを打ち切る．

距離係数の効果

次のようにパラメータを特定してMAMのシミュレーションを行なった．第１に，意見値ごとのクラスタのサイズが小さくなり過ぎる可能性を考慮し，態度値を２次元の２進数で定義する．態度値は 00, 01, 10, 11 の４種類である．第２に，各セルの自己との距離を，Latané, Nowak & Liu (1994) を踏襲して 0.84 と仮定した．

上記の特定の下で態度初期比率と距離係数を要因として(２×６)，条件ごとに２０の run を実行した．態度初期比率は，態度の各次元の初期値が 1 となる確率が p[1]=0.4 か p[1]=0.5 かによって操作する．P=0.4 のとき，態度値の ‘00’，’ 01’，’ 10’，’ 11’ が初期状態で出現する確率はそれぞれ，0.36, 0.24, 0.24, ０．１６ である．距離係数要因は距離係数の n を０～４で1.0刻みに変化させて操作する．また統制条件として，セルの状態が自分自身と隣接する４セルだけで決まる隣接条件を用いる．隣接条件では距離係数の値をnbと表わし，自分との距離を 1.0 とおく．

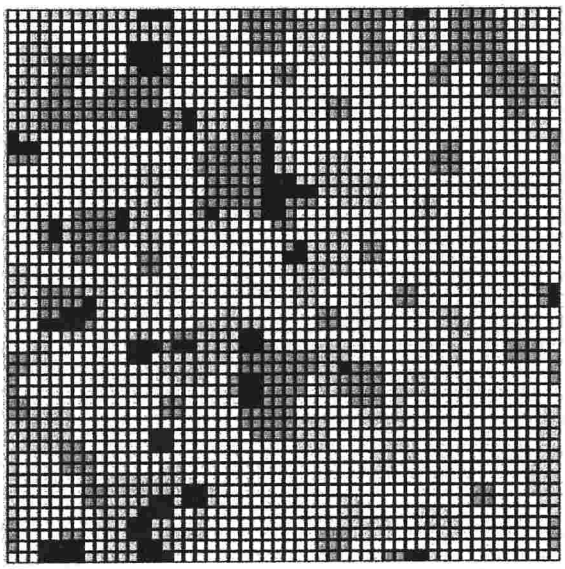


図２：最終ラウンドのセル空間の例

(色が薄い方から濃い順に00, 01, 10, 11を指す．)

ＳＩＭの結果と直接比較できるのは態度値の初期出現率が異なる条件である (p[1]=0.4)．図２はp[1]=0.4 で n=4 のときの最終ラウンドでのセル空間の様子を例示している．ランダムな態度値の分布から出発しながら態度値ごとのクラスタが出現しているのが分かる．

図３は最終ラウンドでの態度値ごとのセル度数の平均値を表す．距離による影響の差が少ない条件 (n=0～2) では初期の最大多数派がほぼ全体を制圧している．逆に，影響が近隣に限定されるほど少数派の残存率が高まる．なお，ＳＩＭの結果と同様に，初期の少数派 (01, 10, 11) は初期比率に比べて，最終の比率は条件にかかわらず低くなっている．初期比率が01や01より小さい11は，減少率が01や01よりも大きい．

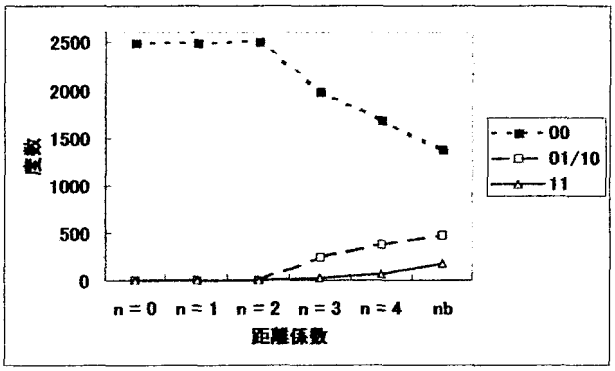


図3：最終ラウンドでの平均セル度数
（‘01/10’ は 01 と 10 の平均である．）

P[1]=0.4 の条件では初期状態で平均 1210 のクラスタが存在していた．ここにクラスタとは，その内部の何れかのセルと隣接関係でつながれたセルの集合を意味する．１セルでも１つのクラスタと数える．このクラスタ数は最終ラウンドではn=0～2 で平均で1.6～2.2に減少する．n=3で35.9，n=4 で85.1，nb で127.6 である．図４は態度値ごとの平均クラスタ数を示す．最大多数派の 00 は度数が高く相互につながっているため，距離係数にかかわらずクラスタ数は少ない．距離が影響力を遮断する条件では，少数派のクラスタ数が増える．すなわち，距離が障壁となるときには少数派は多数の小さいクラスタを形成しやすい．このことは次の図５でも確認できる．図５は態度値ごとの最大クラスタサイズの平均を示す．

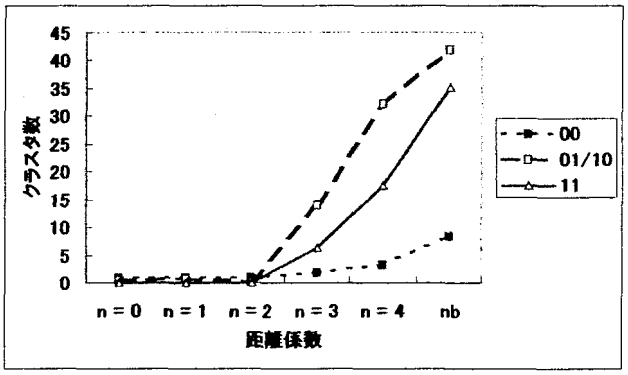


図４：平均クラスタ数

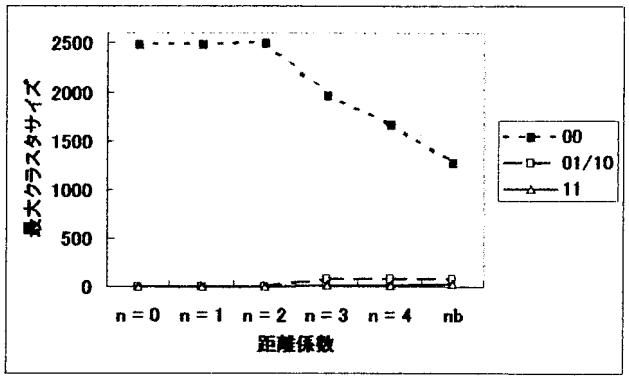


図５：最大クラスタサイズの平均

以上の結果はＳＩＭで観測された２つの傾向がＭＡＭの前提でも確認できることを意味する．つまり第１に態度ごとにクラスタが生じている．第２に，距離の影響が適当な範囲にあることを前提に，初期比率での少数派もクラスタになることによって残存できる．しかし均衡時での少数派サイズは初期比率に比べて減少する．なお，距離による影響の障壁が小さければ少数派が存在できないことは，同様に距離係数を操作すればＳＩＭでも確認できる(高木，2000)．

各態度値の初期出現確率が等しい条件(p[1]=0.5)の結果は次のようだった．距離係数 n が０のときはすべての run で，何れか１つの態度値のクラスタがほぼ全体を支配する．n が１と２のときも基本的には同じことが生じる．が，n＝1 のときは３０％の run で，n＝2 のときは５０％の run で，セル空間は２つの大きなクラスタによって分割されている．n=0～2 でのクラスタ数の平均は２前後である．が，n が３か４，および隣接条件下では，４つの態度値の度数がほぼ等しい状態で均衡する．クラスタ数の平均は135.4(n=3)，134.3(n=4)，192.0(nb)だった．以上の結果も，距離係数が小さいときは全体を統合する傾向が生じることを表している．

隣接セルの構成

セル空間におけるクラスタ化をマイクロなレヴェルでとらえるために，各セルがどの態度のセルと隣接しやすいかを調べてみる．セルは４つのセルと隣接する．態度の初期出現率が異なる条件(p[1]=0.4)でその隣接セルの構成をセルの態度値ごとにまとめたのが図６である．どの態度のセルも生き残る３条件(n=3，n=4，nb)だけを分析の対象とした．図６はその3条件の平均を表す．

図６は次の２点を伝えている．第１はどの態度でも距離の近い態度との隣接比率が高いことである．まず同じ態度のセルとの隣接比率が高い．態度が異なるセルの中では距離が遠い態度ほど隣接比率が低い．つまり距離のある態度とは遠ざかるようにクラスタが組織化されている．第２は隣接比率がセル度数の影響を受けていることである．すなわち，最

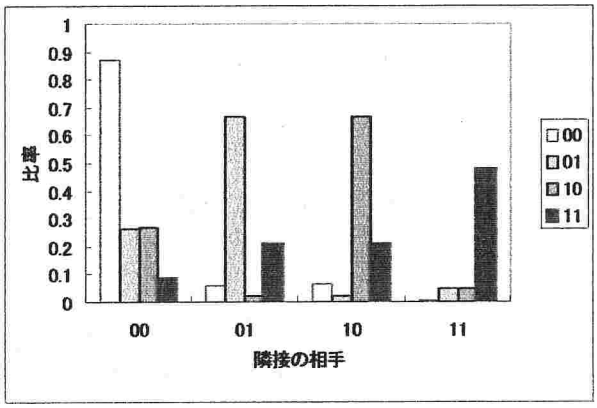


図6:態度間の隣接比率

終ラウンドでの度数の高い態度ほど同じ態度のセルと隣接しやすい．初期比率×距離係数（２×３）別に，４つの態度のそれぞれの度数とその態度の最終度数の相関をとると(n=20)，相関係数の値は .511(p<.01, 両側)～ .971(p<.001)となる．つまり同じ条件の同じ態度であっても，度数が高いほど同類隣接率は高くなる．

上記の第２点は度数分布を前提にした一種の構造効果と考えることができる．度数が高い態度のセルは，偶然からでも同態度のセルと隣接する確率が高くなる．現実の社会でも一般に，サイズが大きい集団ほど内集団との接触可能性は高い（Blau, Blim, & Schwartz, 1982; South, Bonjean, Markham & Corder, 1982）．

上記の構造効果の影響を除いて態度ごとの隣接しやすいさを評価するため，つぎのように「隣接係数」を求める．まず最終ラウンドでの度数分布を前提に隣接セルをランダムに選ぶと仮定したときの，態度ごとの隣接セル数の期待値を求める．隣接する４セルが特定の構成になる確率は多項分布によって求めることができる．この多項分布から隣接セルの態度の出現確率を求め，（態度ｊに対する態度ｉのセルの）隣接係数 r_{ij} を次の式で定義する．

隣接係数 $r_{ij} = \mathrm{Ln}(F_{ij}/E_{ij})$.　　（３）

ただし F_{ij} は態度ｉのセルが態度ｊのセルと実際に隣接した比率，E_{ij} は多項分布から求めた期待比率である．実際の比率と期待比率が等しいとき r_{ij} は０となる．実際の比率の方が大きい［小さい］とき，r_{ij} は正［負］となる．なお，定義上 $r_{ij} = r_{ji}$ である．

図６と同じ３条件での，この隣接比率の平均を表したのが図７である．図７は次の3点を示している．

第１は態度の距離が近い相手との隣接係数が高いことである．どの態度値でも自分と同じ態度のセルとの隣接傾向は期待値を越え，最も遠い態度の相手との隣接傾向は期待値よりはるかに小さい．中間的な距離にある相手への隣接係数はその中間である．この傾向は初期出現比率が等しい条件（p[1]= 0.5）でも確認でききる．

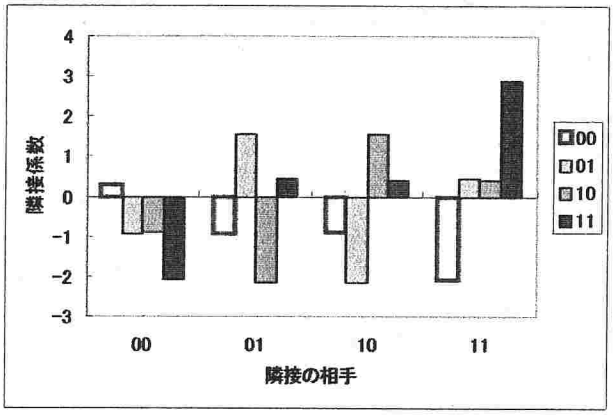


図7：態度ごとの隣接係数

第２は，度数が小さい態度のセルは同じ態度のセルと隣接する比率が低いものの（図６），隣接係数はむしろ高いことである．つまり期待値と比べれば同じ態度のセルと隣接しやすい．この結果は条件別に(n=20)態度ごとの度数と隣接係数の相関をとっても確認できる．相関係数は −.995 ～ −.897 である．つまり度数の小さい態度セルほど度数分布を前提とした期待値より同じ態度のセルと隣接しやすい．

この第２の結果は，多数派集団はゆるく構成され(loosely-knit)，少数派集団は「凝集的に」構成されることを意味している．多数派セルは，図２の 00（白）のセルのように，大きなクラスタを形成しつつも内部に異なった態度のクラスタを含んでしまう．内部の異態度クラスタを外部に押しやれば 00 のセルは同態度セルとの隣接係数を高めることができる．しかしそのようにして隣接係数を高めなくても多数派の 00 は自己を支持する影響源（同態度のセル）を数多く周囲に見出して存続することができる．逆に少数派の態度のセルは相互に接触するようなクラスタを形成してはじめて生き残ることができる．

第3は，最大多数派の 00 では異なった態度のセルとの隣接度数がすべて期待値を下回るのに対し，最小の 11 は 00 よりは距離の近い 01 や 10 との隣接傾向を高めていることである．同じ態度のセルを周囲に見出しにくい 11 は，00 よりは距離の近い 01 や 10 と接近して 00 の影響を遮断するときにはじめて，生き残ることができると解釈できる．ちなみに，４つの態度がほぼ同数の度数で均衡する，初期出現率が等しい条件(p[1]=0.5)では，各態度セルは上記の 00 と同様の隣接係数のパタンを示している．

クラスタ化に及ぼす態度間距離の効果

ＳＩＭとは異なりＭＡＭでは意見間の距離という要素が導入されている．既述の隣接係数の結果は距離の大きい態度同士は遠ざかる（隣接しにくい）ことを示している．

意見間の距離は隣接傾向以外の点でもクラスタ化の様相に影響を与えるだろう．まず考えられるのは，ある態度が多数派となるとき距離の大きい態度が圧力（適合度の低下）に直面し，度数を減らすであろうことである．

図８は p[1]=0.4，n=4 のときのシミュレーション（便宜のためシミュレーション１と

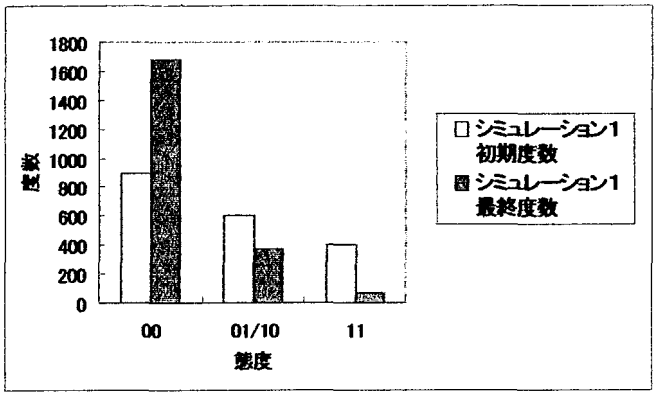


図8:シミュレーション1での態度別度数

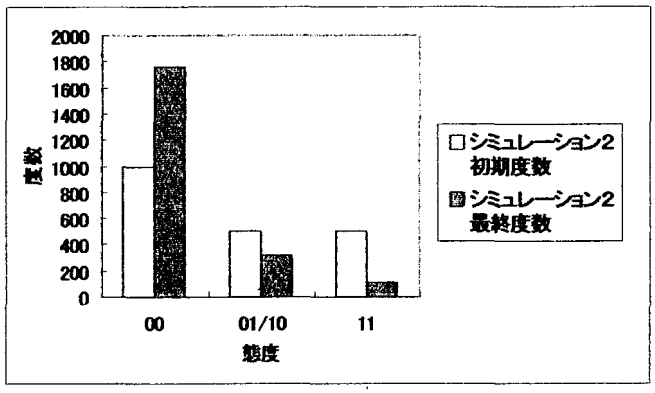


図9:シミュレーション2の態度別度数

呼ぶ）での，各態度の初期度数と最終度数を表す．既述のように初期状態で最大度数をもつ 00 は増加して他の態度は減少する．特に最少数派の 11 の減少率が大きい．11 の度数が落ちる原因の１つは初期度数が01や10より小さいことである．しかし増加する00との距離が大きいことも原因かも知れない．このことを確かめるために，00，01，10，11の初期出現率を２：１：１：１にして，n=4 でシミュレーション２を実施した．結果を図９に示す．初期出現率が01,10と同じでも，増大する00との距離が大きい11は，01と10の平均よりもはるかに最終度数が小さい(対応ｔ検定，df=19，p<.0001)．

意見が２つしかなく意見間の距離も想定していないＳＩＭでは，初期状態での少数派は最終的にはより小さくなるだけだった．しかしより多くの態度間で距離を想定するＭＡＭでは，初期状態での構成によっては少数派が減少しないことも予想できる．例えば距離の遠い態度がともに多数派になるような場合である．このとき，距離の遠い態度が相互に抑制し合うため，少数派が生き残る可能性は高まるだろう．

そこで 00，01，10，11の初期出現率を３：１：１：３， n=4 としてシミュレーション３を実施した．結果を図１０に示す．シミュレーション３では，初期少数派の 01 と10は減少するどころかむしろ増加している（対応ｔ検定，両側でそれぞれ p<.005，p<.05)．注意

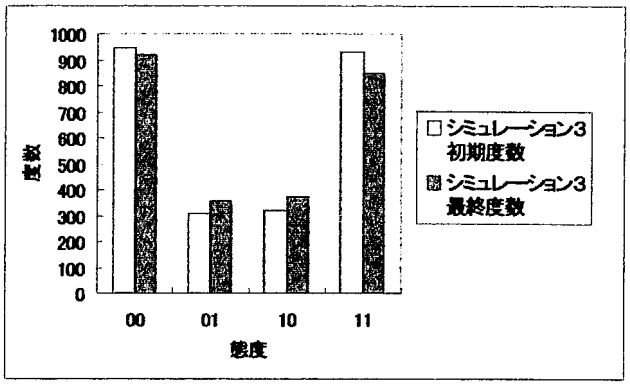


図10:シミュレーション3の態度別度数

すべきはこのときの00と11はともに，シミュレーション１の00に匹敵する初期出現率を持ち，01と10はシミュレーション１で著しく減少した11よりも初期出現率が低かったことである．

図９の結果は拮抗する態度間の距離が大きいときに限って生じる．初期出現率が１：１：３：３で，初期状態で大きいのが距離が小さい10と11であるなら，初期少数派の00と01はほとんど消滅してしまう．

考察

本研究は態度を多次元的に定義するMAMのシミュレーション結果を検討した．結果は次のようにまとめることができる．

第１に，このモデルによるセル空間でのクラスタ化は距離係数，つまり距離に応じて影響力が及ぶ程度の影響を受ける．セル間で大域的に影響が及ぶ場合は１つの態度が全体を支配することが多い．各態度が一定のクラスタとして空間内で存続できるのは，逆に遠くの影響源からの影響を遮断できる場合である．この結果はＳＩＭにおいても確認できる(高木，2000)．

第２に，クラスタ化は初期の態度分布に依存的であり，初期に多数派だった態度がより多数に，初期に少数派だった態度がより少数になる傾向を伴う．この傾向は初期度数分布に基づく構造効果と見ることができる．この傾向もまたＳＩＭでも確認されてきた点である．

第3に，しかしMAMが予測するクラスタ化では態度間の距離の要因が作用する．距離が遠い態度のセルは相互の適合度をより低下させるため，相互に遠ざかるようにクラスタ化が生じる結果になる．そしてある態度が多数派に成長するときには距離の遠い態度のセルが度数を減らしやすい．逆にかけ離れた２つの態度が相対的に多数である状況では，両者が牽制し合うため，両者と態度が比較的近いセルの少数派クラスタが存続し，あるいは成長することがある．

本研究のシミュレーション結果は態度に基づく集団化に対して一定の 含意を持っている．MAMが予測する均衡状態では多数派の集団と少数派の集団が異なった構造を持つ，とい

う点である．多数派となる集団のエージェントは近隣に類似した態度のエージェントを多数見出すことができるので，同類と接触する傾向をあまり高めずにクラスタとして生き残る．その結果，空間的には自己の集団の「内側」に異なった態度の集団を位置させる．反対に少数派は，比率の上では同類と隣接することが少ないにもかかわらず，確率的期待値から比べれば同類と隣接傾向を高めたクラスタとして生き残る．また，態度が比較的近い少数派と近づく傾向もある．重要なのは，こうした傾向が，「少数派ほど同類を好む」といった個人レヴェルの選好を介することなく、構造的な淘汰の結果として生じ得ることである．

態度間の距離がもたらす効果については，本研究では可能な状況の一部を検討したに過ぎない．他の状況でいかなる帰結が導かれるかについては体系的に検討する余地がある．

引用文献

Blau, P.M. (1977). A macrosociological theory of social structure. *American Journal of Sociology, 83,* 26-54.

Blau, P.M., Blim, T.C. & Schwartz, J.E. (1982). Heterogeneity and intermarriage. *American Sociological Review, 47,* 45-62.

Epstein, J.M. & Axtell, R. (1996). *Growing Artificial Societies.* Cambridge:The MIT Press.

Latané, B., Nowak, A. & Liu, J.H. (1994) Measuring emergent social phenomena: Dynamism, polarization, and clustering as order parameters of social systems. *Behavioral Science, 39,* 1-24.

Mayhew, B.H. & Levinger, R.L. (1976). Size and density of interaction in human aggregates. *American Journal of Sociology, 82,* 86-110.

Nowak, A. & Latané, B. (1994). Simulating the emergence of social order from individual behaviour. In N. Gilbert & J. Doran (Eds.) *Simulating Societies.* London: UCL Press, Pp. 63-84.

Nowak, A., Szamrei, J. & Latané, B. (1990). From private attitude to public opinion: A dynamic theory of social impact. *Psychological Review, 97,* 362-376.

Sakoda, J.M. (1971). The checkboard model of social interaction. *Journal of Mathematical Sociology, 1,* 119-132.

Schelling, T.C. (1971). Dynamic Models of Segregation. *Journal of Mathematical Sociology, 1,* 143-186.

South, S.J., Bonjean, C.M., Markham, W.T. & Corder, J. (1982). Social Structure and intergroup interaction. *American Sociological Review, 47,* 587-599.

高木英至 (2000). Social Impact シミュレーションのタネと仕掛け．『日本グループ・ダイナミックス学会　第４８回大会　発表論文集』(予定).

Ⅲ．１．文化の成立に関する試論：交換規範はいかに創発するか

高木英至（TAKAGI, Eiji）
（埼玉大学　教養学部）

キーワード：社会的交換、規範、進化ゲーム、コンピュータシミュレーション

【要約】社会の中で、規範などの文化的要素は人間に外在的、拘束的な規制的作用を及ぼす。本稿の課題はこうした文化要素がなぜ存在するか、成立するに至るか、という一般的な問題から発している。この課題を広義の「交換事態」（分配を含む）における規範について考察する。本稿の視点は文化要素が進化ゲームにおける戦略進化として理解できる、というものである。本稿ではこのアイディアを計算実験において試す。まず分配正義（例：平等）が戦略進化として説明できることを述べる。次いで等量交換を結果するような互酬規範の文化が出現できることを示す。

問題の背景

説明対象としての社会的事実

「社会的事実（social facts）」とは集団に基因して個人に外在的・拘束的に作用する、個人からは独立した実体を指す(e.g., Durkheim, 1895)。この概念は個人心理から独立した社会学固有の研究対象として Dulkheim が提起した概念であることは、よく知られている。ただし社会的事実の概念は、社会の流れ、制度、集合的感情など、Durkheim の著作を通して眺めてみても多様な対象を指しており、その意味内容に必ずしも一貫性はない。

筆者なりに再解釈すれば、社会的事実は大別して次の２つを指すと言えるだろう。

第１に、社会的事実は社会全体の状況そのものを指している(e.g., Calhoun, Light & Keller, 1994:4-5)。この意味での社会的事実はほとんど、社会構造ないし社会秩序と区別する所がない。社会全体の状況が個人に外在的、拘束的に作用することは次の定式で表現できる。いま、個人の行動は合理的選択に従うと仮定しよう。このとき、個人 i の効用関数 u_i は次のように言い換えてもよい。

$$u_i(a_1, a_2, \ldots, a_i, \ldots, a_n) = u_i(a_i, o_i).$$

仮定から、個人 i は u_i を最大化するように自らの行為 a_i を選ぶだろう。このとき、社会全体の状況とは諸個人の行為の総体、つまり $\mathbf{a}=(a_1, a_2, \ldots, a_j, \ldots, a_n)$ と表せる。$\mathbf{a}$ から a_i だけを除いた行為のヴェクトルを o_i で表わしてもよい。もし社会の規模が十分大き

く、個人が o_i を動かせない（所与とおく）なら、ちょうど非協力ゲームにおけるプレイヤーの選択行動のように、個人は o_i を前提にして最適な a_i を選ばざるを得ない。このように社会の状況（o_i）は個人に外在し、かつ個人を拘束する。

社会的事実の第２の解釈可能性は、社会状況 **a** からも独立し、**a** を規制するような情報性のパタンと考えることである。この意味での社会的事実は規範、価値観、制度など、通常は文化の構成要素と見られる制御的な情報である。もし一定の社会に一般的なシンボルや知識まで何らかの意味で「規制的」な社会的事実と考えるなら、この第２の意味での社会的事実は文化と言い換えて差し支えない。

社会的事実を以上のように考えたとき、社会諸科学の研究の中で取り残されているのは第２の意味における社会的事実の成立過程の解明である。規範を例にとろう。社会心理学の中で最も研究されてきたのは、規範の中でも広い意味での交換事態に働く規範(justice)である。図式的に言えば、社会心理学は衡平理論（Equity Theory, e.g., Adams, 1965）以来、justice が破られたときに人がどのように反応するかを研究してきた(e.g., Tylor et al., 1997)。どのような種類の justice があるかが議論されたことはあるものの (e.g., Deutsch, 1975)、その justice 自体の成立は問題にされることはほとんどなかったと言ってよい。

進化ゲームの適用

筆者の基本的立場は進化ゲーム論と表現することができる(e.g., 山岸, 1998)。すなわち、エージェントがそれぞれの戦略に従って行為すると仮定したとき、一定の社会状態は戦略の進化に基づく動的な均衡としてその存在が説明できる、と考える立場である。この進化ゲーム論の自然な拡張は、説明対象とする均衡的な社会状態が上記の社会的事実を含んでいる、と考えることである。

この基本的立場に発する本稿の問題関心は、第２の意味での社会的事実の成立、特に規範の成立の説明に進化ゲームの論理を適用することにある。ここで進化ゲームがどのように文化の成立を説明できるかを議論してみよう。

進化ゲームは複数のマイクロな行為主体（以下、エージェントと呼ぶ）の均衡点を見つけるための用具である。従ってマイクロ経済学や通常のゲーム理論で適用される均衡モデルと異なる分析を提示する訳ではない。わずかな違いは、エージェントの行動原理が最適化（将来の利得の最大化）ではなく進化的な適応（結果として生じる適応度の向上）であること(Axerlod, 1997)、および進化ゲームが形式上は異なった均衡の定義に従っていることである。いずれの均衡分析でも、その課題は、均衡するエージェントの行為の組 $\mathbf{a}=(a_1, a_2, \ldots, a_i, \ldots, a_n)$ を見出すことにある。つまり、その課題は第１の意味での社会的事実を説明することにある。

このように見ると、第２の意味における社会的事実つまり文化は第１の社会的事実と切り離すことはできない。広く用いられる定義では、文化とは学習し伝達される価値、規範、知識、シンボルを指している(Calhoun, et al., 1994:7, 52)。こうした文化の要素は何ら

かの意味で個々のエージェントの行為(a_i)の中に埋め込まれていなければならない。もしそうでなければ学習も伝達もできないからである。以上は、進化ゲームを含む均衡分析が同時に文化の成立を説明するモデルであり得ることを意味している。

より正確に言えば、進化ゲームによる文化の成立の説明は次のような形式に従うことになるだろう。まず、進化ゲームによって説明される $\mathbf{a}=(a_1, a_2, \ldots, a_i, \ldots, a_n)$ とは、行為（言わば表現型）の組ではなく、各エージェントが選択した戦略（遺伝子型）の組である。文化要素は個々の戦略 a_i の要素（パラメータ）として表現されることになるだろう。もし考慮したエージェントの均衡下の戦略のパラメータに共通のパタンがあれば、そのパタンが進化した文化であると言えるはずである。例えば「資源を与える」という戦略の中に「平等に与える」ことを指定するパラメータが共通に見出せるなら、当の集団の中に「平等主義文化」が見出せたことになる。

むろん進化ゲームの射程に入る文化の説明には次の限定がつくことも見逃せない。進化ゲームは適応度に基づく説明形式である。従って、進化ゲームによって説明できるのはエージェントの適応度に応じて戦略の選択が生じる場合に限定されることである。例えばエージェント間の相互影響によってエージェントの態度ないし文化要素の収斂を説明するモデルがある(Nowak & Latane, 1994;Axelrod, 1997, chap. 7)。こうしたモデルはエージェントの適応度に論及することなく「文化」の成立を説明しており、適応度に基づく現象とは別の文化の側面をモデル化していると言うべきだろう。ただしこの相互影響に基づく説明にも背景では適応度の要素を含んでいる可能性がある。その意味では相互影響による文化成立モデルと進化ゲームによるモデルの間の関係も追究されるべきかも知れない。

交換に基づく規範の成立

文化の要素の中でも特にその成立に議論を要するのが規範である。ここで規範とは、何らかの命令的（injunctive）要素を持つ、つまりそれに従うことが直接的には当人の利益を害するものの、従うことを社会的に（他のエージェントから）要請されるルールを指すと解しておこう。こうした命令的なルールはサンクション、言い換えれば選択的な誘因(selective incentives)が欠如しては維持できない。そのため、規範の成立の説明には選択的誘因メカニズムの成立を説明する課題が付随することになる。

進化ゲーム論から規範の成立を説明する直接的な方法は、Axelrod(1997, chap. 3) のように、逸脱への科罰などの選択的誘因の提供を含む戦略の進化を説明することである。この場合、選択的誘因の維持に伴う２次的ディレンマを解決できるような戦略の進化可能性を見出すことが課題となる。

しかし直感的に言えば、Axerlod(1997) が描くような、逸脱への科罰を伴う規範が広範に成立し社会を制御してきたとは、想像し難い。人間社会を規範が頻繁に規制したとすれば、より「単純な」選択的メカニズムが存在して来たとしても不思議はない。

筆者の想定では、選択的なメカニズムの基礎となって来たのが広い意味での社会的交換

である。ここでは社会的交換に、限定交換、一般交換（高木，1994）だけでなく分配まで含めて考えてみよう。これら社会的交換は諸個人の活動成果をプールし、リスクに対処する機構であったと考えることができる。そのため、人間および人間社会の存続はたぶんに社会的交換に依存して来たと考えることができる。社会的交換の存在が普遍的であるのもそれ故と言えるだろう。重要なのは、社会的交換が何らかの意味でその便益圏を限る点である。資源のやり取りを制御する結果として、社会的交換はそれ自体が選択的誘因機構として機能できる。

一般交換を例に説明しよう(Takagi，1996)。一般交換とは一定の集団内で成員が任意の成員に利他的になる（資源を与える）ことであると定義する。つまり一般交換は、集団内での利他性と同義である。しかし一般交換は、無条件的に利他的になる戦略（普遍的利他戦略）からは実現しない。一般交換を持続させる戦略は利他的になるエージェントにのみ利他的になるという選択性が必要であり、さらに利己戦略の中で安定的に進化するためには、利他的である「善き市民」に対してだけ利他的であるという、高度の選択性が要求される。安定的な一般交換圏とは、つまり利他性の範囲を「善き市民」のクラブ（一般交換クラブ）に限定してはじめて成り立つと言える。そのため一般交換は、それ自体高度に選択的な誘因要素を埋め込んだ戦略によって成り立つと考えることができる。

一般交換が持つ選択的誘因機能はさらに次のようにして集団を基盤とした規範へと拡張できる(Takagi，1999)。社会が一般交換クラブとして成り立つ状態を仮定しよう。このクラブの成員は相互に、善き市民であることを求めている。もしここで、このクラブ内で公共財の供給なり共有資源の管理なりの、共通の利益となる事案があったとしよう。例えば灌漑用水などの公共財の供給が集団的な課題になったとしよう。このとき、成員の中に、「善き市民」の要件としてこの公共財に出資することを含める戦略を思いつくエージェントが生じても不思議はない。この新たな戦略は、一般交換と「公共財への出資」を結びつけることによって、公共財の供給を促進することになる。Takagi(1999) のシミュレーションは、このような拡張された戦略が進化し得ることを示している。重要な点は、この場合、「公共財に出資する」という新たな規範のための選択的誘因機構として一般交換が作用する点である。むろんこのような規範が機能できるのは、一般交換が成り立つ圏内に限定されることになる。

社会的交換が普遍的であり、かつ交換に内在する選択性が容易に他の集団事案とリンクできるとすれば、人間社会が限定された圏内で規範を有効に機能させることができたのは社会的交換のためであるかも知れない。社会的交換は社会に創発的に備わる社会的事実を組織化する位置にある可能性がある。

本稿の課題

以下で論じるのは社会的事実一般や規範一般の創発ではない。その一般的な問題の限定的課題として、交換規範（社会的交換に成り立つ規範）の一部がどのように創発するかに

ついて素描することである。社会的交換はそれ自体として社会秩序の重要な一角をなす。同時に交換規範は社会規範の根幹をなすはずである。

交換規範が適用できる状況には大別して２種類がある。第１は協力の成果を個人間で「分配」する場合である。第２は、各自が所有する財を相互に与え合うような（狭義の）社会的交換である。２者間で完結する「限定交換」は一般交換とともに狭義の社会的交換の典型をなしている。以下ではまず分配に成り立つ規範の成立に関する筆者のシミュレーション結果に触れ、次いで最も重要な交換規範の１つ、限定交換における「互酬規範」の成立に関するシミュレーション結果を述べる。

分配規範：衡平と平等

この節では、分配がいかなるルール（規範）に基づくかを、コンピュータシミュレーションを用いて分析する。内容は高木(1998)、Takagi(1999) に基づく。本稿では前節で記した問題関心を例示する程度の記述にとどめる。

協同－分配モデル

シミュレーションは次のようなモデルとして構成される。エージェントが集団をなして狩猟（漁猟でもよい）をすると考える。狩猟での貢献水準はその主体の資源である。獲物が集団の成果である。集団が大きければ資源も大きくなり、集団の成果も増大する。問題はその成果にどのいうな分配が生じるかである。

この分配の事態でいかなる分配ルールが出現するか？　少し考えただけでも多くの要因が絡むことが予想できる。結果の一般性を保証することを考慮し、モデルには次の要因を導入する。

第１の要因は資源がエージェント間で均一か格差があるか、である。均一なら平等以外の分配ルールの実現は難しそうに見える。格差があるなら、過去の実験結果(e.g., Pruitt, 1972)が示すように、貧者は平等を主張し富者は衡平に傾くだろう。

第２は資源保有の不確実性である。不確実なら（そしてエージェントが危険回避的な効用関数を持つなら)、不確実性が低減するように、より平等に近い原則が選好されるだろう。

第３は、別の原則を掲げる他の集団がエージェントによって選択可能かどうかという点である。選択可能なら、つまり移動が自由なら、富者はより富者優遇的なルールを持つ集団に移動し、貧者は逆の動きをするに違いない。その結果として社会全体にいかなる秩序が生じるかは思考実験の価値がある。戦略的進化はこの移動の自由を前提にしている。

第４に、エージェントの貢献「率」も考慮に入れる必要がある。エージェントは手抜き、つまり保有する資源を手元に残すことができる。平等は貧者にとって快い原則かも知れない。だが周知のように平等原則は社会的ディレンマ（free-rider 問題)をひき起こす。その結果、平等集団では社会的手抜きが生じやすく、集団の成果は低減するだろう。この予

想通りなら、貧者も最終的には、見込みのない平等集団を離れる可能性がある。衡平など高貢献者を優遇するルールは、富者優遇的であると同時に手抜きを罰する特質を持っている。

第５に、集団間の競争の有無も考慮する。例えば範囲が限定されたフィールドで複数集団が狩猟をするとすれば、獲物は限られているので競争が生じるだろう。この競争は、直感的には、より貢献する者をひきつけるルールの進化を促すだろう。逆に、各狩猟集団の前に新たなフロンティアが広がっているなら、集団間の競争は生じない。競争の有無でルールの帰趨は異なるかも知れないのである。

以下の本稿では、集団間での成果の分配と集団内での分配に、Amnon Rapoport らの集団間競争事態の分配規則定式（Rapoport & Amaldoss, 1997）を適用する。Rapoport らの定式とは、集団 i の成果を次の f に比例して成員 k に分配するという定式である。

$$f = X_{ki}^{c} / X_{i}^{c}, \quad (\ 0 \leqq c) \qquad (1)$$

ただし　X_{ki} ：集団 i の成員 k の貢献量

$X_{i}^{c} = \Sigma X_{ki}^{c}$.

c を「集団内競争係数」と呼ぶ。Rapoport らのこの定式の特徴は、分配ルールを１つのパラメータ c によって表現する点である。c = 0 のとき、分配規則は平等(Equality)を意味する。貢献量にかかわらず成員は等しい成果を受け取る。c = 1.0 なら衡平(Equity)である。貢献と成果は比例する。c > 1.0 の場合は「超メリトクラシー(Super-merit system)」とでも呼び得る。ちょうどプロスポーツチームのように、より貢献する者を優遇し貢献しない者を排除するルールとなる。(ただし以下では、分配に適用したときの不合理さを考慮し、通常の演算規則に反して X_{ki} = 0.0 のとき X_{ki}^{0} = 0.0 と定義した。)

（１）と同様の定式を集団間競争にも当てはめる。（１）の c に相当する係数 g を「集団間競争係数」と呼ぶ。集団間競争があるとき、g = 0.0 なら、集団が成果を獲得する機会は集団の合計貢献量にかかわらず等しい。g = 1.0 なら成果の獲得機会と集団の貢献総量とは比例する。g > 1.0 なら、最大資源量を誇る集団の成果獲得機会が非常に高い。

以上の協同－分配モデルを次のようなシミュレーションモデルに翻訳した。

シミュレーションモデル

概略　３００のエージェントからなる仮想社会を想定する。エージェントは自己の戦略に従って行動する。

３００のエージェントは初期保有の資源量に応じて１００名ずつの３つの階層（富者／中間／貧者）に別れる。具体的な資源量は条件によって異なる。ただし全主体の資源量の総計は条件にかかわらず 一定 (600.0)　である。

戦略は２つの要素からなる。第１の要素は集団所属である。社会には初期状態で９つの

集団がある。９つの集団は集団内競争係数が異なる。0.0 ～ 4.0 まで、0.5 刻みに、この係数が異なった集団が存在している。集団に属さない主体は１０番目の名目的な集団に属する、と考える。第２の要素は自己の保有資源のうち貢献にあてる比率である。貢献（にあてる）比率は 0.0 ～ 1.0 の間の、0.25 刻みの５水準がある。集団の資源とは成員が貢献した資源の総和である。

シミュレーションは離散的なラウンドで進行する。各エージェントはラウンドごとに利得を得る。１ラウンドの構成の仕方は下記の２つのモデルで相違する。 エージェント間の戦略の分布は後述の方法でラウンド進行とともに変化する。

競争モデル　シミュレーションモデルには競争モデルと非競争モデルがある。競争モデルは集団間に成果獲得をめぐる競争関係を前提にする。集団間競争係数の値がその競争関係の強さを表す。

競争モデルでは１ラウンドが１００００回の試行からなる。１回の試行で１単位の成果が何れかの参入者のものとなる。参入者とは集団、もしくは個人で競争に参加する（１０番目の集団に属する）エージェントである。１試行での参入者の成果獲得確率は g による集団間競争の定式に基づく。参入者が集団のとき、その成員への成果の分配はそれぞれの集団の集団内競争係数で決まる。貢献総量の大きな集団は資源獲得の確率が高い。個人参入者は、成果獲得確率は低いものの、成果を一人占めできる。

１ラウンドの１００００試行は１００個の「１００試行ごとの試行群」からなる。試行群ごとに各エージェントの利得を計算し、１ラウンドの利得は１００個の利得の合計で定義する。試行群ごとに利得を計算するのは、危険回避的効用関数を仮定したとき（後述）に集団所属による危険回避の効果が生じるようにするためである。

このモデルでは２種類の不確実性が存在する。第１は要因として導入する資源保有の不確実性である（後述）。第２は成果の獲得が確率的であることによる不確実性である。第１の不確実性が無である条件でも第２の不確実性は存在する。したがって効用関数が危険回避的なら集団への所属にはリスクプレミアム分の利益が生じる。

各試行群での利得計算では、各主体が得た成果（の分け前）の量に 10.0 をかけ、さらに手元に残した（貢献しなかった）資源量を加算する。つまりこのモデルでは、成果獲得に見込みがない場合は資源を手元に残した方が得をする。

非競争モデル　成果獲得において参入者間の競争が生じないようにするため、次の計算手順を用いた。参入者（集団および個人参入者）ごとに１００００回の試行を行う。各試行において１単位の成果を獲得する確率は、その参入者の貢献資源総量／全主体の資源総量(=600)である。ただしエージェントとその資源は有限であるので、貢献能力の高いエージェントを獲得する競争関係は参入者間に存在している。他の点では競争モデルと非競争モデルに差はない。

戦略分布の変化　主体の戦略の変化には２つの源泉がある。第１は戦略の学習である。各エージェントは自己が属する資源階層内の他の成員の戦略と結果を観察していると仮定す

る。階層ごとに、ラウンドの利得の上位５名の戦略が下位５名に代入される。つまり失敗者は成功者の戦略を学習すると仮定する。第２の源泉は突然変異である。戦略の２要素のそれぞれはラウンドごとに他の値にランダムに変化する(p=0.01)。

要因計画　両モデルに次の３つの要因を導入してシミュレーションを実施した。①危険選好（中立的／回避的）：危険回避条件では利得に対数変換[Ln(利得+1)]を施す。②資源格差（大／中／小／無）：富者、中間、貧者の各階層の資源は、各試行でそれぞれ以下である。(3.9,2.0,0.1),(3.0,2.0,1.0),(2.5,2.0,1.5),(2.0,2.0,2.0)。③資源不安定性（大／小／無）：無の条件では主体の資源量は変化しない。小条件では、主体は各試行群において、0.5 の確率で自己の資源階層の資源量を得、0.25 ずつの確率で他の階層の資源量を得る。大条件では主体は３階層の資源量を等確率で得る。つまり大条件では階層間の資源格差は実質的にはない。「資源保有の不確実性」とは、資源格差と資源不安定性の増加関数だといえる。

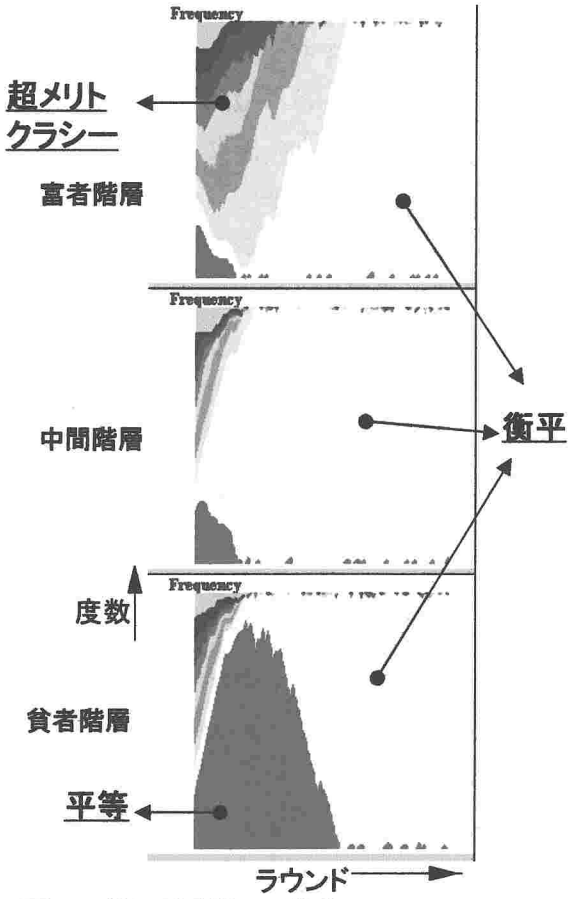


図1:分配規範の進化

競争モデルではさらに集団間競争係数を５水準の要因として導入した（g = 0.0/0.5/1.0/2.0/4.0)。

競争モデルでは計１２０、非競争モデルでは２４の条件で、条件ごとに１０の run のシミュレーションを実施した。

Ｒｕｎはすべて１００ラウンドで打ち切った。どのＲｕｎでもほぼ、戦略の分布は収束している。

シミュレーション結果

要因数が多いため、各要因の効果（の検定)、水準別の結果、などを説明すると煩雑になる。ここでは次のように概略の結果を述べておこう。競争モデルと非競争モデルも結果においてさほど相違はないので、両モデルを一緒にして結果を述べる。

（１）エージェントの選好が危険中立的であるとき、成果獲得のための集団化は（富者階層を除いて）生じない。集団化が生じるとすれば、集団間競争が強い条件下での超メリト

クラシーである。集団化が生じやすくなる条件はエージェントが危険回避傾向を持つことである。

（2）次の条件が揃うとき超メリトクラシーが優越しやすい。(a) エージェントが危険回避的でない（上記）。(b) 資源格差が小さい。(c) 集団間競争が強い。

（3）衡平ルール（貢献に比例した成果の配分）は次の条件が揃うときに生じやすい。(a) 危険回避傾向がある。(b) ある程度の資源格差がある。(c)資源保有に不確実性がある。

（4）平等ルール（貢献にかかわらず等しく成果を配分）は、衡平が生じやすくなる条件に加え、資源保有の不確実性がさらに強くなったときにだけ生じる。

（5）平等集団では社会的手抜き（低貢献率）を招くために、多くの条件で優越的になれない。

図1は衡平が全階層を通じて支配的になったＲｕｎにおける集団所属戦略の進化過程を例示している。この例では、当初、富者階層で超メリトクラシーが、習慣階層で衡平が、貧者階層で平等が優越する。分配ルールと階層とのこの関係は、広範な条件で観測され、過去の知見（e.g., Komorita & Chertkoff, 1973; Pruitt, 1972）を再現すると考えられる。しかし図から明らかなように、まず貧者階層で平等が衡平にとって替わられる。このことは平等クラブが専ら貧者によって構成されるようになるため、貧者は「搾取」する富者や中間層を内部に見出せなくなり、また社会的手抜きに基づく非効率性のために、成果を獲得できずに沈んで行く。他方、超メリトクラシー集団も富者ばかりで構成されるようになるため、内部は貢献量の等しい平等集団になり、同様に「搾取」する貧者や中間層を見出せずに富者にとっての有利さも消滅してしまう。その間に中間層で栄えた衡平集団が富者と貧者を統合して最終的に優越している。

考察

上記の分配規範のシミュレーションは、条件（モデルのパラメータ）に応じて異なった分配ルールが生成されることを示している。この結果は分配ルールにまつわる１つの謎を理解する鍵になるだろう。その謎とは、社会に異なった結果を含意する別の分配ルールが存在し、同一人物が関与する社会の別の局面で異なった分配ルールが適用され得る、という点である。このシミュレーション結果からも実は同じことが予測できる。例えば集団間競争が重要になるビジネスの局面ではメリトクラシーが支配的になり、リスク回避が重要な家族や仲間集団では平等が優越する、などである。

上記のシミュレーションで最も顕著な作用を及ぼしたのはエージェントに働く資源保有の不確実性だった。単純な表現をすれば、どの分配ルールが優越するかは、主に、不確実性の度合いによって決まる。不確実性が強くなるに従い、分配ルールは衡平、さらには平等へと傾いて行く。この結果は、平等主義がリスクへの対処として生じるという従来の知見(e.g., Cashdan, 1980, 1985)に符合すると言えるだろう。

だがこのシミュレーション結果には不満がある。実際の社会の日常観察からすれば、実

際の分配規範は平等と衡平の中間点にあることが多いだろう。少なくとも平等は、最も自明な「分配解」である。にもかかわらず、上記のシミュレーションでは平等が優越するのは極端に不確実性が高い条件だけだった。

平等は一方でリスクへの対処という有益な機能を果たす。にもかかわらずこのシミュレーションで平等ルールが弱かった原因は、社会的手抜きを招きやすく、平等集団が成果をもたらせないことにある。だとすれば、何らかの仕掛けによって社会的手抜きを回避する戦略が、平等主義とともに進化することがあるだろう。

筆者が考えたその仕掛けとは、内集団志向に基づく選択的誘因である。エージェントの戦略次元に「仲間基準」s (0.0-1.0) という戦略次元を追加した。例えば s が 0.5 のエージェントは、貢献率が 0.5 未満であるか、0.5 未満の人を追放しようとしない成員を「黒い羊」と定義し、追放しようとする（追放を決意する成員が過半数なら追放がおこる)、と仮定した。この戦略次元は、戦略が集団への貢献に応じた選択的誘因を形作る余地を与えている。

仲間基準を導入したシミュレーション結果は、低貢献の成員を潜在的には追放する傾向の進化を示すものだった。同時に、この選択的誘因（追放可能性）によって、平等ルールはより生じやすくなることを見出した。

この追加的なシミュレーション結果は、平等主義的な文化が、追放を内包した強い内集団志向とともに進化することを示唆すると言えるだろう。

互酬規範の成立

2エージェント間の相互的な資源のやり取りを「限定交換(restricted exchange)」と呼ぶ。限定交換は古典的な交換理論(e.g., Blau, 1965)が社会的交換の典型例と想定する交換である。同様に、限定交換を対象とする互酬規範も、交換規範の代表と言える。この節ではこの互酬規範がなぜ創発されるのかを考察する。

互酬規範

分配規範とともに交換規範としてよく論及されるのが「互酬規範(The norm of reciprocity)」である(Gouldner, 1960)。互酬規範とは、自分に恩恵（被害）を与えた者には返礼することを指令する。互酬規範が社会生活のいろんな局面で成り立つことは古くから人類学や社会学で取り上げられてきた。

互酬規範の普遍的な成立に強力な論拠を与えて来たのが Axelrod (1984) や数理生物学者による、囚人のジレンマ（ＰＤ）への進化ゲームの適用結果だろう。「協力」を「相手に資源を与えること」、「非協力」を与えないことと考えれば、ＰＤは典型的な（２者間の）社会的交換のモデルとなる。エージェント間でＰＤが成り立つときにいかなる戦略が進化するかに関する現時点での結論は正確には複雑である。しかし協力志向のエージェントに

対して協力する（社会的交換を行う）ような戦略が進化することは大筋の結論と考えてよい。この結論を前提にすれば、互酬規範は文化として成立する事象と見られているものの、その文化事象は進化ゲームの論理で出現すると推論することができる。

しかし自分への協力者に協力することは互酬規範の１つの側面に過ぎない。協力する（資源を与える）か否かだけでなく、社会的交換では一般に「どの程度与えるか」が問題になる。そして互酬規範には「相手がくれたのと等量の資源を相手に与える」というニュアンスが含まれている。この等量性は、多くを受け取れば多くを返さなければならないという規則性として表現されている(e.g., Pruitt, 1968)。社会的交換におけるこの「等量性」の存在はどのように説明することができるだろうか？　以下のシミュレーションで検討するのはこの「等量性」の出現可能性である。

シミュレーションモデル

１００のエージェントからなる社会のモデルを用いる。エージェント間に Giving Game を仮定する(高木, 1994; Takagi, 1996)。Giving Game では、各エージェントは一定量の資源を持ち、その資源を相手に分け与えることができる。１世代が複数の試行からなる。Giving Game を用いた限定交換のシミュレーションの中で筆者が以前に用いた戦略のうち、非協力戦略（常に誰にも与えない）とＴＦＴ戦略だけが選択可能と仮定する。ＴＦＴ戦略とは、自分に資源をくれる相手には与えるような戦略であり、概ね次のように挙動する。(1)各世代の初期の試行では自発的に他のエージェントに資源を与える。(2) 自分に資源をくれる相手を「仲間」と認定する。(3) 仲間には毎試行、一定量の資源を与える。しかし２度の試行で裏切った（資源をくれなかった）相手は２度と仲間とは認めない。言うまでもなくＴＦＴ戦略は、ＰＤにおけるＴＦＴを念頭に置いた戦略である。

この２つの戦略だけを用いたとき、１００エージェントの集団の中に５ほどのＴＦＴエージェントが存在すれば、容易に他の非協力エージェントを駆逐できることが分かっている。が、この知見は、ＴＦＴが他エージェントに与える資源量を固定した場合の結果である。以下のシミュレーションでは与える資源量が可変的と仮定している。

各エージェントの戦略には、資源の与え方（非協力／ＴＦＴ）以外に、最大供与量（他者に与える最大量）と最小受領量（自分が他者から受け取ったと考える最小の資源量）という、２つの追加的戦略次元がある。最小受領量（閾値）に満たない量の資源をもらっても、エージェントは「受け取った」とは考えない。他者の最大供与量と最小受領量がエージェントには分かる、と仮定する。さらに、エージェントは相手の最小受領量しかその相手には与えないと仮定する。また、自分の最大供与量を超える最小受領量を持つ相手は交換の対象から外すと考える。最大供与量と最小受領量は４水準（１～４）からなり、２つの２値変数で表現している。

このモデルはＴＦＴが与える資源量を固定した場合とは次の２点で異なっている。第１は、ＴＦＴエージェント同士であっても２つの追加的戦略次元の値によっては交換が生じ得ないことがある。第２に、与え合う資源量が異なる（片方が過大／過小に受け取る）交換が理論上は可能なことである。

エージェントの戦略次元は初期状態で無作為化した。シミュレーションの１つのＲｕｎの流れは図２に示す。１つのＲｕｎは５００世代からなる。各世代の終りに「進化」が生じ、低利得エージェントの戦略が高利得エージェントの戦略に置き換わる（他のエージェントは次の世代も同じ戦略を継承する）。同時に戦略各次元には一定の確率で突然変異が起こる。１世代は２００試行からなり、その世代でのエージェントの利得は各試行での利得の時間割引合計で表現される。

シミュレーションは２要因計画（３×３）に基づく。第１の要因は初期比率要因（各世代の初期状態でのＴＦＴエージェントの比率）であり、３水準を設ける（５％／１０％／５０％）。第２は資源量要因（各エージェントが各試行で保有する資源量）であり、同様に３水準を設定する（４／１０／２０）。９つの条件の各々で１０のＲｕｎを繰り返した。

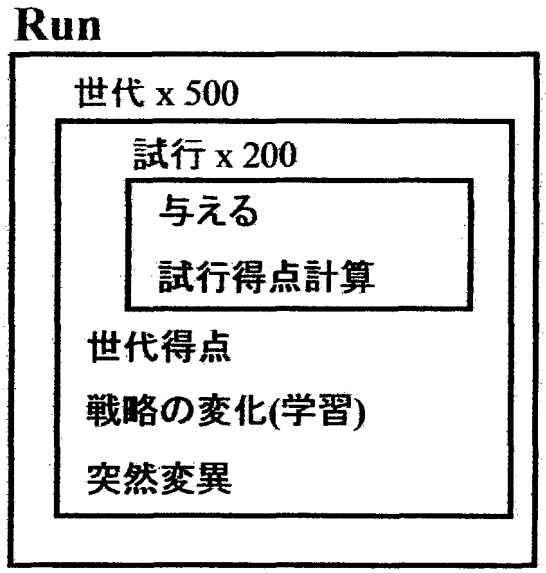


図2:シミュレーションのRunの構造

結果

以下、主に最終世代での結果を示す。

ＴＦＴの優越　条件ごとのＴＦＴ比率は９６～９９％（全体平均は９７．８％）である。ＴＦＴが全体をほぼ制圧している。有意な効果を持ったのは資源量要因だけであり（*F*検定）、資源量が４のときにＴＦＴの度数が低く、最大供与量と最小受領量は小さい（*SNK*検定）。

交換戦略の進化　最大供与量と最小受領量のエージェント度数を、全条件を通してプールした結果を図３に示す。円の面積が度数を示している。図３では、最大供与量と最小受領量という２つの戦略次元の値が一致した状態で落ち着くエージェントが多いことが読み取れる。

Ｒｕｎごとの最大供与量と最小受領量の分析から次の点が分かる。

（1）最大供与量と最小受領量はともに偏って分布している。ＴＦＴエージェントだけを取り出し、その２次元の値の各々の分布から適合度検定統計量とと同様のカイ二乗を算出すると、最大供与量でも最小受領量でも、有意水準(*df=3*)を大きく超える値が得られる（全Ｒｕｎの平均値はそれぞれ、74.0 と 118.4)。

（2）最大供与量と最小受領量は一致する傾向がある。９条件中のどの条件でもこの２次

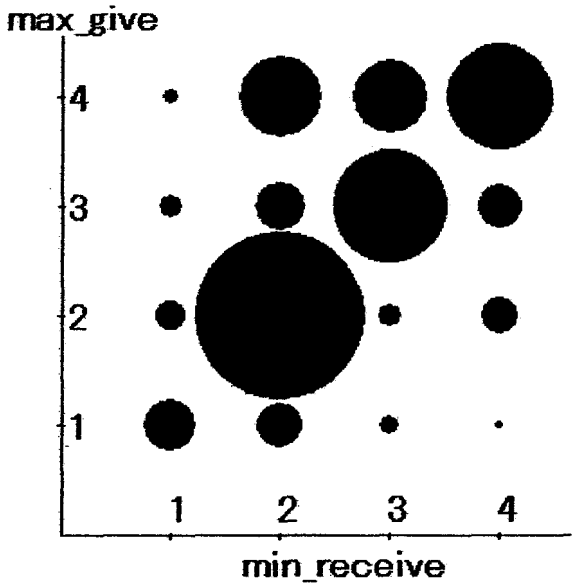


図3:最大供与量と最小受領量の度数

元の値は一致していることが多い。一致率は全体平均で７１．６％である。ただしこの一致率の大小は周辺分布に依存する。周辺度数の影響を除去するために、次の式で両次元の一致の程度を指標化し、その指標をＲｕｎごとに算出した。

一致指標　＝*log*(一致度数／一致期待度数)

ただし「一致期待度数」とは両次元の独立を仮定したときに期待される一致度数である。この指標がゼロなら実際の一致が両次元独立の場合と等しい。正なら偶然以上に両次元の値は一致していることになる。どの条件でもこの指標の平均は有意に正である（*t=3.7*～*24.4, df=9*)。

戦略ごとの棲み分け　ここまでの結果からは次のことが分かる。第１に、このシミュレーションでは戦略進化が生じて最大供与量と最小受領量がＲｕｎごとに偏る。第２はその２

つの量が一致する傾向があることである。個々のＲｕｎを眺めると最小受領量の最大グループ（多くは値２）ともう１つのグループ（多くは３か４）ができている。つまり、多くのＲｕｎで異なった２つのエージェントクラスタが仮想社会の中で共存していることになる。

次の分析から、Ｒｕｎの中で共存する集団の内部で交換関係が生じやすいことが分かる。つまり集団が付き合いの上で棲み分けしている（同類づきあい傾向）。

この同類づきあい傾向を数値で得るために次のようにして同類づきあい指標を求めることにした。まず最小受領量の値の周辺分布から、最小受領量が同じエージェント間で生じる交換関係数期待値E_W（無作為の関係形成を仮定ときの交換関係数の期待値）を求める。さらに、実際に各次元で同じ値のエージェント間で生じた交換関係数F_Wを求める。ここに「交換関係」とは、戦略におけるパラメータの値が、相互に、定常的に資源を与え続けるようになっているエージェントのペアを指す。ここで同類づきあい指標を次のように定義する。

同類づきあい指標 ＝ $log(F_W/E_W)$

同じ最小受領量の値のエージェント間の交換関係数が偶然から予期できる程度であれば、同類づきあい指標はゼロである。同類づきあいの程度が偶然以上であればこの指標は正になる。同類づきあい指標はどの条件でも有意にゼロより大きい（$t=15.1\sim31.4$, $df=9$）。つまり交換関係の多く（全体平均で85.8%）は最小受領量が同じエージェント間で生じている。前提から、最小受領量が同じエージェント間では等量の交換が生じている。

同類づきあい傾向は最大供与量次元について同じ計算をしても確認できる。しかしＲｕｎごとの同類づきあい指標はすべての条件で、最小受領量の方が有意に高くなる（対応 t検定）。つまりエージェント間の棲み分けは最大供与量より最小受領量に応じて生じていることになる。

考察

エージェント間の社会的交換を許すシミュレーションにおいて、「どれほどまでを与えるか」、および「どれほどもらえば『受け取った』ことにするか」の基準の進化が観測できた。この進化は後者の基準についてより明確だった。つまりこれらの基準についてエージェントの社会の中で「文化」が発生したといえる。個々の社会の中では複数の基準（いわば下位文化）が共存しがちであったが、交換関係は同一「文化」を共有するエージェント間で生じやすかった。同じ「文化」の中では、互酬規範が指示するように、等量交換の傾向が進化している。

結び

社会的事実や文化を進化ゲーム論から説明するという視点に立ち、シミュレーション結果から、分配における平等や衡平といった分配ルール、限定交換における互酬的な等量性ルールが生じ得ることを本稿で述べた。このシミュレーション結果は、よく知られた交換規範が進化ゲームの論理から生成され得ることを示している。注意すべきはこれらのルールが社会的交換事態を前提に作動していることである。つまり、これらのルールへの違反は社会的交換による便益からの排除を意味している。それゆえ、これらのルールは選択的誘因に裏づけられた規範として成立していると言ってよいだろう。

文化をはじめとする社会的事実は２重の意味で「所与」である。第１に、社会的事実は個人にとっては所与のものとして提起された。さらに第２に、社会的事実は研究者にとっても所与のものと扱われ、その解明、特に成立の解明が求められることが稀だった。本稿が提起した進化ゲームの視点はもとより、社会的事実のあらゆる側面の出現を目指すものでは、ない。しかしこの視点は社会的事実や文化の成立の重要な側面を説明できるものと、筆者は見込んでいる。

引用文献

Adams, J.S. (1965) Inequity in social exchange. *Advances in Experimental Social Psychology, 2,* 267-299.

Axelrod, R. (1984) *The Evolution of Cooperation.* New York: Basic Books. アクセルロッド　松田裕之(訳)　『つきあい方の科学』、1987、ＨＢＪ出版局.

Axelrod, R. (1997) *The Complexity of Cooperation.* Princeton, NJ:Princeton Univ. Press.

Calhoun, C., Light, D., & Keller, S. (1994) *Sociology* (6th edition), New York: McGraw-Hill.

Cashdan, E.A. (1980) Egalitarianism among hunters and gatherers. *American*

Anthropologists, 82, 116-120.

Cashdan, E.A. (1985) Coping with risk: Reciprocity among the Basarwa of Northern Botswana. *Man (N.S.), 20,* 454-474.

Deutsch, M. (1975) Equity, equality, and need. *Journal of Social Issues, 31,* 137-149.

Durkheim, E. (1895) Les regles de la methode sociologique. Paris: Press Universitaires de France. デュルケーム 佐々木交賢（訳） 『社会学的方法論』、1973、学文社(ただし訳は1950年版に基づく).

Gouldner, A. (1960) The norm of reciprocity. *American Sociological Review, 25,* 161-178.

Komorita, S.S. & Chertkoff, J.M. (1973) A bargaining theory of coalition formation. *Psychological Review, 80,* 149-162.

Nowak, A. & Latane, B. (1994). Simulating the emergence of social order from individual behaviour. In N. Gilbert & J. Doran (Eds.) *Simulating Societies.* London: UCL Press, Pp. 63-84.

Pruitt, D.G. (1968) Reciprocity and credit building in a laboratory dyad. *Journal of Personality and Social Psychology, 8,* 143-147.

Pruitt, D.G. (1972) Methods for resolving differences of interest. *Journal of Social Issues, 28,* 133-154.

Rapoport, A. & Amaldoss, W. (1997) Comparison of different rules for winning contests and distributing public goods in between-group competitions. Paper presented at the 7th International Social Dilemma Conference.

高木英至 (1994) 社会的交換のシミュレーション・パラダイム. 埼玉大学紀要、30、23-55.

Takagi, E. (1996) The generalized exchange perspective on the evolution of altruism. In W.B.G. Liebrand, & D.M. Messick, (Eds.), *Frontiers in Social Dilemmas Research.* Berlin: Springer, Pp. 311-336.

高木英至 (1998) 「分配正義の生成： シミュレーションによる分析」、『日本グループ・ダイナミックス学会 第４６回大会 発表論文集』、28-31.

Takagi, E. (1999) A simulation analysis of the emergence of distributive justice. Paper presented at the 8th International Conference on Social Dilemmas at Zichron Yaakov, Israel.

Tyler, T.R., Boeckmann, R.J., Smith, H.J., & Huo, Y.J. (1997) *Social Justice in a Diverse Society.* Boulder: Westview.

山岸俊男 (1998) 『信頼の構造』、東京大学出版会.

Ⅲ. 2. 利他性

高木英至
(埼玉大学　教養学部)

この小論のテーマは利他性の進化である。利他性（Altruism）とは自己を犠牲にして他者を利すること、難しくいえば自己の適応度を低めて他の個体の適応度を高めることを意味する。命を賭して人に尽くすことも、気軽に人に親切にすることも、利他性の意味する範囲である。

■人は情の下で立つ

このことわざは人が互いの思いやりの中で生きていくことを表している。社会は思いやりで成り立つことといい換えてもよい。テーマである利他性は他者を思いやる行動特性を指し、この「情」こそは社会心理学者が利他動機と呼ぶものに対応している。

人が何らかの利他性を有することはよほどシニカルな人でない限り認めるところである。社会はまた人の心と双対的であるから、利他性は社会の中にもその痕跡をとどめている。社会学の古典的な用語にゲマインシャフトとゲゼルシャフトという言葉がある（テンニース)。そのゲマインシャフトとは血縁、地縁、友情などで結びついた、いわば利他的な社会圏を指す。社会のすべての局面が利他的ではないけれども、利他性を特徴とする部分を第一次的な環境としながら人が生きていることは確かである。

むろん、人の利他動機が真の利他動機であるかどうかを疑うことはできる。利他行動が「ノーベル賞でももらうつもりで頑張っている」ことの現れである場合もあるだろう。また、われわれが「思いやり」と呼ぶものの中には、実は「社会的交換」として受取る「思いやり」もある。人に何かをしてあげることを前提に相手から「思いやり」をもらう、という場合である。このような直接的な交換で自分の利益が増すのは自明であるから、この種の「思いやり」は利他性の発現ではない。

別の見方で動機の利他性を疑うこともできる。社会心理学には「真の利他動機」があるかどうか、という議論がある。例えば、人が窮状にある人を見かねて助けてあげたとしよう。その行動が窮状にあった相手に本当に共感して（相手の苦しい感情を取得して）助けた結果なら、真の利他行動といえる。が、助けた人は窮状にあった人の存在を不快と思い、その自分の不快を低減するために困った人を助けたのだとすれば、その利他行動は自己利益（不快低減）追求の結果であり、真の利他動

機に基づくとはいえない、という議論である。ここで問題になるのは、相手への共感に基づく真の利他動機が本当にあるのか、ということだった。実験結果による一応の結論は 'Yes' である。

■情けは人のためならず

ヒトの利他性がなぜ進化したのかは１つのなぞといえる。利他性は自己の適応度を低めて他者を助けることであるのに、進化で生き残る遺伝子とは適応度を高めるものであるはずだからである。

利他性が存在しにくいことは次のように例示できる。いま、利他的な個体だけからなる社会と利己的個体だけからなる社会があるとしよう。利他的な社会では誰もが皆に親切にするから、その成員の適応度は利己的社会の成員の適応度を上回る。だから利他的な社会はその個体とともに生き残りやすい、という議論はもっともらしく見えるけれど、成り立たないのである。もし利他的社会のある成員が突然変異的に利己的になったとしよう。このとき、利他的社会の利己的成員は周囲の利他的成員の恩恵を受ける上に、利己的であることによって自分の資源も浪費しない。だから適応度は特に高くなる。そこで利己的個体が繁殖によって自分のコピィをより多く生み出すか、利他的成員が利己主義へと改宗するかして、利他的社会はじきに利己主義が支配する社会になってしまうからである。

利他性がこの世に出現する可能性を見出す鍵は上のことわざにある。情は直接的には自己を犠牲にして人のためにするものである。しかしそうした犠牲的行為がめぐりめぐって自分の適応度を高める、つまり人のためではないという結果になるような連関があれば、利他性はこの世の中で進化できるはずである。

利他性を出現させる連関が血縁者間で存在することを見出したのが、ハミルトンの有名な包括適応度の考えである。この考えは、一定の条件の下で、遺伝子が近い血縁者に利他的になる遺伝子が進化的に有利であることを導いている。血縁者に利他的になる遺伝子を持つ個体は、その利他性によって自己の適応度は低めるものの、自己と近い遺伝子を持つ血縁者の適応度を高めるために、結果として同じ遺伝子を増やすことになる、という考えである。この考えと整合的に、あるコンピュータシミュレーションによる研究は、子供に対する親の利他性が、親に接近しようとする子供の側の傾向とともに進化できることをデモンストレイトしている。こうした血縁的利他性は現代の人間社会でも観察できる。例えば、調査回答者は悩みごとの相談のネットワークの中に多くの血縁者を含めているし、多くの援助は血縁が近い者から得ているのである。

問題は非血縁者間で、あるいは個体間の血縁性を仮定せずに利他性が出現することを説明できるか、という点である。社会心理学は血縁者の枠を越えて利他行動が生じ得ることを確認して来たし、血縁以外を契機としたゲマインシャフトは頻繁に

観察されて来た。

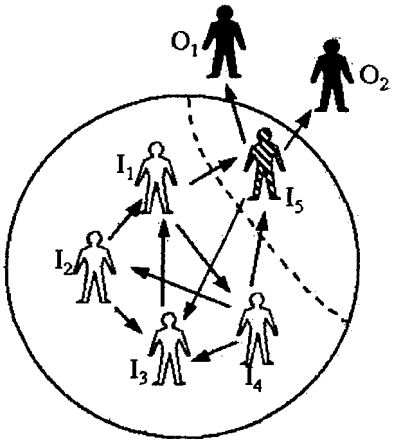
血縁性の仮定なしに利他性を出現を説明する試みの１つは「一般交換」の視点に基づく議論である。一般交換とは、ある社会圏の任意の成員間で、お返しによらない利他行動が生じる状態を指す。この視点は、個人の側の利他性を社会状態としての一般交換に置き換えて考える。私が行なったのは次のような一般交換のシミュレーションである。一般交換戦略と利己戦略（誰にも与えない）を考える。一般交換戦略にはいろんなヴァリエーションがあり、自分の資源をどれだけ他者に与えるか、どのような他者を利他行動（資源を与える）の対象とするかが戦略によって変異する。さらに成功した（利得を多く得る）戦略が増えると仮定したとき、利他的な者にだけ恩恵を施すような一般交換戦略が利己戦略を淘汰し、計算上の社会で利他性を実現できることをシミュレーション結果は示した。

血縁に基づく場合も基づかない場合も、利他性の出現は上記のことわざの原則に従っていると考えられる。直接的には無償の情（利他行動）が、人のためではなく自分のためになるという関連を作り出せる限りで、利他性は出現できる、ということである。

■万人の友人は誰の友人でもない

英語では上のようなことわざがあるらしい。ある人の友になりたければ誰に対しても良い顔はできない、という意味だろう。実は似たような原則が上記の一般交換シミュレーション結果にも現れている。

上記のシミュレーションで出現した利他性は普遍的利他性、つまり誰にも無条件に利他的になること、ではない。利他性（一般交換）が出現するためにはまず、利己的な者は利他性の対象から排除しなければならない。が、それだけではない。利己的な者に対し利他的になる者も排除しないと、安定的に利他性は出現できないのである。



今、図のように、I1～I5の利他主義者とO1, O2の利己主義者がいるとしよう。さらに利他主義者I5は利己主義者にも利他的になるとしよう。このとき、利他性の恩恵はI5を通して利己主義者にも放出されてしまう。I5は心の広い利他主義者であるけれど、利他性のサークルから見れば「放蕩息子」である。もしこの放蕩息子が増えてしまえば、利己主義は利他性の恩恵を受けて増殖し、逆に利他主義者は減じて、結果として利他性が崩壊する危険がある。ここで安定的に利他性が成長するためにはI5のような放蕩息子を利他性サークルから排除する必要が生まれる。つまり、利他性の原資に上限がある限り「閉じた利他性のサークル」を作り、その内部だけで利他性を完結させなければ、利他性が頑健に進化することができない、ということである。この含意は「あらゆる利他性は差別的利他性である」というG．ハーディンの説に一定の支持を与えている。

■金持ち喧嘩せず

社会的機能からいえば利他性の存在は資源の不確実性への緩衝、いわば保険の役割を果たす。各人の持つ資源は貯蔵が効かず（例えばある種の食糧）、明日資源を得られるか否かが高度に不確実であるとしよう。このとき、利他性は資源を得られぬ不運な者が生き残ることを可能にする。人間（社会）の存続に利他性の進化が不可欠だった所以である。

しばしば誤解があるのは、資源保有の不確実性を資源水準の低さ（貧しさ）と取り違えることである。貧しいから人は助け合うような気もするけれど、貧しさ自体が利他性を生じさせやすくすると考える理論的根拠はない。

逆に、社会が全般に貧しければ利他性は生じにくくなるだろう。利他性は社会の資源をプールし、成員間で再配分することを意味する。もし再配分によって生き残る（子孫を残す）ことができるならば、利他性が存在する意味はある。しかし全般に資源が欠乏し平等に配分しても誰も生き残れないほど世の中が貧しいなら、競争や闘争を経て、一部の者だけが資源を得て生き残るような社会秩序が生まれるだろう。利他性はある意味で「金持ち喧嘩せず」の帰結ともいえるのである。

Ⅲ．３．協力呼びかけのシミュレーション：安心は信頼を低下させる

高木英至
（埼玉大学　教養学部）

キーワード：協力、社会的ディレンマ、選択的受け容れ、信頼感

【要約】この研究は協力関係への選択的受け容れ（selective inclusion）メカニズムが進化的な均衡として成り立つか否かをコンピュータシミュレーションによって検討する。１００のエージェントの各々が順番にN人協力関係への呼びかけを行い、胴元と協力者に利得が生じるようなシミュレーションモデルを構成した。協力関係ができれば胴元と協力者には利得が生じる。しかし胴元・協力者は違反をすれば、違反者は他の関与者の犠牲の下に利益できる、という状況を想定した。シミュレーション結果は、大勢では選択的受け容れがエージェントの戦略として進化し、協力関係が生じやすくなることを示した。さらに、社会全体の協力水準は一定であっても、違反への誘因が高い方が、未知の他者への信頼（trust）が高まる、という効果が観測された。後者の結果は「安心が信頼を壊す」という議論を論理的に支持する結果であるかも知れない。

はじめに　－　選択的な受入れと排除

社会的ディレンマ研究において、ディレンマの解決、すなわち非効率的なナッシュ均衡解（非協力状態）への潜在的傾向を克服することは、一貫して課題であり続けた(e.g., Foddy, Smithson, Hogg, & Schneider, 1999)。その解決法についていろんな議論が積み上げられてきたことはいうまでもない。

協力状態を実現するために実際に人々が行っている方法の１つは、協力しそうな人だけを受入れ、非協力的な人を排除して協力関係を作ることだろう。こうした選択的な受入れ／排除（selective inclusion/exclusion）には少なくとも次の２つの意味がある。第１は、人はその動機づけ志向に応じて、一貫して協力的、あるいは非協力的な傾向を示すことがあることである(e.g., Kramer, McClintock, & Messick, 1986; Liebrand, & van Run, 1985)。多くの人がそうした一貫性を持つと見込めるの

なら、協力するに決まっている人だけを予め受入れ、しからざる人を排除した方が、協力性の出現という点では合理的であるに決まっている。第２の意味は、協力関係に入れることがエージェントの利得達成に寄与する限り、選択的な受入れ／排除が協力に向けての誘因システムとして機能し、結果としてエージェントの協力を促せることである。

しかし次の点はまだ不明と言わなければならない。すなわち、選択的な受入れ／排除が戦略として進化できるのか、言い方を変えると、選択的な戦略にもとに協力が生じやすくなるという状態が（進化的な）均衡として達成可能なのか、という問題である。この研究では、以上の問題関心の下に、「協力呼びかけゲーム」を前提とした計算実験を行おうとする。

シミュレーションモデル

協力呼びかけゲームの構造の概要

シミュレーションの背景となるゲーム構造は次のような状況である。

(1) エージェント数が１００の社会を仮定する。この社会では空間を仮定せず、エージェント間の距離も定義しない。

(2) エージェントの１人が交互に胴元(manager)になって協力を呼びかける。胴元と協力者への利得は協力者数に依存する。利得は胴元の方が協力者より高い。以下に示すように、ゲームではＮ（胴元＋協力者）＝９で胴元・協力者の利得は最大となるように設定する。胴元１人でも利得になるが、Ｎ＝１では利得は低い。

(3) 協力者も胴元も協力関係において「違反」できると仮定する。違反したとき、違反者には追加的利益が生じ、他の参加者には損害が出る。

(4) 上記の(2)、(3)から、この協力呼びかけゲームは利得が非対称的な、Ｎ人ディレンマ構造になっていることが分かる。

利得構造

(1) 胴元が呼びかけた協力関係の当事者数をN_tとするとき、胴元の利益は$a \cdot N_t^{1/2}$で定義する。ただし、（自分を含めて）当事者を１人増やすごとに、胴元にはｂの経費が生じると仮定する。従って胴元の利益U_mは次の式で表せる。

$$U_m = a \cdot N_t^{1/2} - b \cdot N_t \qquad [1]$$

ここでａ＝３、ｂ＝１／２とおく。［１］を微分すれば分かるように、ａ＝３、ｂ＝１／２であるなら、U_m は $N_t = 9$ のときに最大になる。従って合理的なエージェントは、自分が胴元になったときには８人の協力者を集めようとするはずである。

(2) 協力者の利益Ｕｃはａ・$N_t^{1/2}$／１０とおく。胴元にはコスト関数がかかってい

るので最適なN_tを持つけれども、協力者にとっては協力関係の当事者は多ければ多いほどよいことになる。

(3) 協力関係の中で違反者が１人生じるごとに、違反者を含めてすべての協力関係当事者には $-a \cdot N_t^{1/2}/5$ の損害が生じると仮定する。違反者総数がN_dであれば、各参加者が蒙る被害は$-N_d \cdot a \cdot N_t^{1/2}/5$である。もし１人の違反者が出れば、協力者の利得はマイナスになる。つまり違反者が出るような関係には入らない方が得をする。ただし違反者は、自分以外の当事者のそれぞれから $w \cdot a \cdot N_t^{1/2}/10$ の利得を得ることができる。wは違反誘因の強さを表す。また、協力関係が胴元１人であるときは、自分で違反しても違反は成立しない（単に$U_m = a \cdot N_t^{1/2} - b \cdot N_t = a - b$を得る）。

エージェントの戦略

(1) エージェントの戦略は、胴元になったときの「呼びかけ戦略」（７次元）、胴元からの呼びかけへの「受諾戦略」（５次元）、「違反戦略」（３次元）、「賄賂戦略」（２次元）、「信頼(trust)」（２次元）からなる。戦略は計１９次元の２値変数のヴェクトルとして定義する。

(2) 呼びかけ戦略は次のごとく構成する。

(a) 呼びかける人数（０～１５人）、４次元。

(b) スイッチ次元（呼びかける相手をランダムに選ぶか、違反率依存的に選ぶか？）、１次元。

(c) 許容する違反水準（過去の違反率がその水準以下なら呼びかける相手に含める）、２次元で、0.0/0.33/0.67/1.0 の４水準の値をとり得る。 上記スイッチ次元が１のときだけ、その値がエージェントの行動を規定する。

(3) 胴元から呼びかけられたエージェントはその呼びかけを受けるか否かを、自己の受諾戦略に従って判断する。受諾戦略は次のごとく定義する。

(a) スイッチ次元（呼びかけに、確率的に応じるか、違反率依存的に応じるか？）、１次元。

(b) 確率的に応じる場合の受諾確率、２次元。0.0/0.33/0.67/1.0 の４水準の値をとり、上記スイッチ次元が０のときに適用される。

(c) 許容する違反水準（同時に呼びかけられた顔ぶれの過去の違反率から推定される、その関係で違反が生じる確率が、その水準以下なら呼びかけに応じる）、２次元。同様に 0.0/0.33/0.67/1.0 の４水準の値をとり、上記スイッチ次元が１のときに適用される。なお、「その関係で違反が生じる確率」とは、胴元および他の呼びかけられたエージェントの各々の「違反しない確率」の積をｐとしたとき（ｐは１人も違反しない確率を表す）、１－ｐで定義する。

(4) 違反戦略は協力関係に入ったときの違反確率を表す。３次元からなり、１／７刻

みで［0.0, 1.0］の値をとる。
(5) 賄賂戦略は２次元の戦略要素である。第１次元の値が１のときには胴元に「賄賂」（贈賄）の申し出をする。第２次元の値が１のときには、自分が胴元になって賄賂の申し出を受けたときには賄賂を受け取る。賄賂の意味は後述する。
(6) trust＝過去の違反率が分からない相手に推定する非違反率、２次元。0.0/0.33/0.67/1.0 の４水準の値をとる。

賄賂オプション

このシミュレーションモデルでは条件によって、エージェントは「賄賂」の授受をすることができる。賄賂の意味は次のごとくである。あるエージェントが次に胴元になるという情報がランダムに選んだ２０のエージェントに事前に漏れると仮定する。その情報を知ったエージェントのうち、次に違反をするつもりのエージェントは、もし賄賂戦略の第１次元が１なら、次期胴元に賄賂を送る約束を申し出る。また、次期胴元のエージェントの賄賂戦略第２次元の値が１なら、その申し出を受ける。賄賂が生じるとき、贈賄エージェントは違反をし、同時に胴元に成功報酬として $1.5 \cdot w \cdot a \cdot N_t^{1/2} / 5$ の利得を移転する。つまり賄賂を受けた胴元は、贈賄エージェントが違反をしたとしても損をすることはない。贈賄エージェントもカモにする相手が多ければ、胴元に贈賄しても多くの利益を得ることができる。この賄賂オプションは違反を促進する方向で作用するはずである。

エージェントの記憶

このシミュレーションのエージェントは他エージェントの違反率から、そのエージェントを判断する。ここでは reputation は生じないと仮定し、エージェントは自分が関与した協力関係の当事者の行動だけを観測して記憶し、その当事者の違反率を推定すると考える。推定した違反率には（エージェント（A）が過去に観測したそのエージェント（B）の違反度数／AがBを観測した度数）を当てる。もし「AがBを観測した度数」がゼロなら、AはBの違反率に「1.0－Aの信頼」を当てる。また、エージェントの記憶は世代を超えて継承されることはない。

シミュレーションの流れ

シミュレーションの１つのＲｕｎは多数の世代から構成される。以下の分析では２００世代までを観測した。１世代は２００のラウンドからなり、各ラウンドでは１００のエージェントの各々が順番に胴元になる。胴元になる順番はラウンドごとにランダムにに決める。従って１世代の間に２万の協力関係の成立機会が生じる。エージェントの利得は２万回の機会で得た利得の単純な総和である。世代の終了時には次の手順で戦略の変化が生じる。

戦略変化

世代の終了時に戦略の変化（進化）が生じる。戦略変化は交叉（Crossover）と突然変異からなる。

(1) 交叉は次の手順で定義する。世代の終了時に、エージェントを世代での利得の順番に並べ、上位１０％（１０エージェント）と下位１０％を指定する。同順位があればランダムに１０エージェントまでを選ぶ。次いで、上位１０エージェントの中から重複を許して、利得の大きさに比例して２エージェント（ＡとＢ）を選ぶ。さらに交叉位置（１～１８）をランダムに選び、ＡとＢの戦略を交叉させ、重複を許さずランダムに選んだ下位２エージェントにそれぞれの戦略を代入する。すべての下位エージェントの戦略は上位エージェントの交叉を施した戦略に置き換わる。

(2) 突然変異は、交叉の操作後のすべてのエージェントの各戦略次元を確率 0.015 で別の値に変化させることで実施する。

シミュレーションの構造

概略すればこのシミュレーションは次のような構造を持っている。各エージェントは胴元になる等しい機会を持つ。胴元は最適数（$N_t = 9$）までは協力関係を拡大するのが有利であり、胴元にならないエージェントはなるべく多くの協力関係に含まれること、しかもなるべく規模の大きい協力関係に参加することが有利になる。しかしこの状況では違反への誘因が存在する。違反への誘因の強さを決めるのは２つの要因である。第１の要因は、違反したときの誘因を規定するｗである。ｗの値が高い方が違反への誘因は強い。第２の要因は賄賂オプションの存在であり、賄賂オプションが導入された場合の方が全般に違反への誘因は高いはずである。そうした違反への誘因の存在にもかかわらず、呼びかけ・受諾における受入れ／排除の戦略要素の存在が協力関係の成立に貢献するかどうかが、このシミュレーションの焦点になる。

実験計画

今回のシミュレーションでは２×４の要因計画でＲｕｎを実施した。第１の要因は賄賂オプションの有無である（２水準)。第２の要因は誘因の強さ（ｗ）の４水準（小[w=0.1]／中[2/3]／大[1.0]／特大[5/3]）である。８条件の各々で１０のＲｕｎを実施した。事前の試行ではエージェント社会の協力状況の体制は５０世代くらいで固まるので、各Ｒｕｎとも２００世代で打ち切った。各世代の初期状態では、各エージェントの戦略の値は無作為化した。

シミュレーション結果

概況

シミュレーション結果は大勢として、エージェント社会内で協力関係が進化したことを示している。図１は誘因の強さ（ｗ）が大、賄賂ありの条件の第１Ｒｕｎの結果（１－１００世代）を例として示している。まず戦略は、初期状態の無作為化された状態から、すぐに選択的受入れ（呼びかけ・受諾）の状態へと変化する。その戦略の変化に伴い、協力率（当事者１人の関係を除く協力率）が上昇し、上限(100%)に近づいて行く。協力者の平均規模（協力人数）は最初低下するものの、協力率が一定の段階に達すると上昇に転じ、最適人数（９）に近い状態で推移している。同時に信頼も上限近くまで達している。

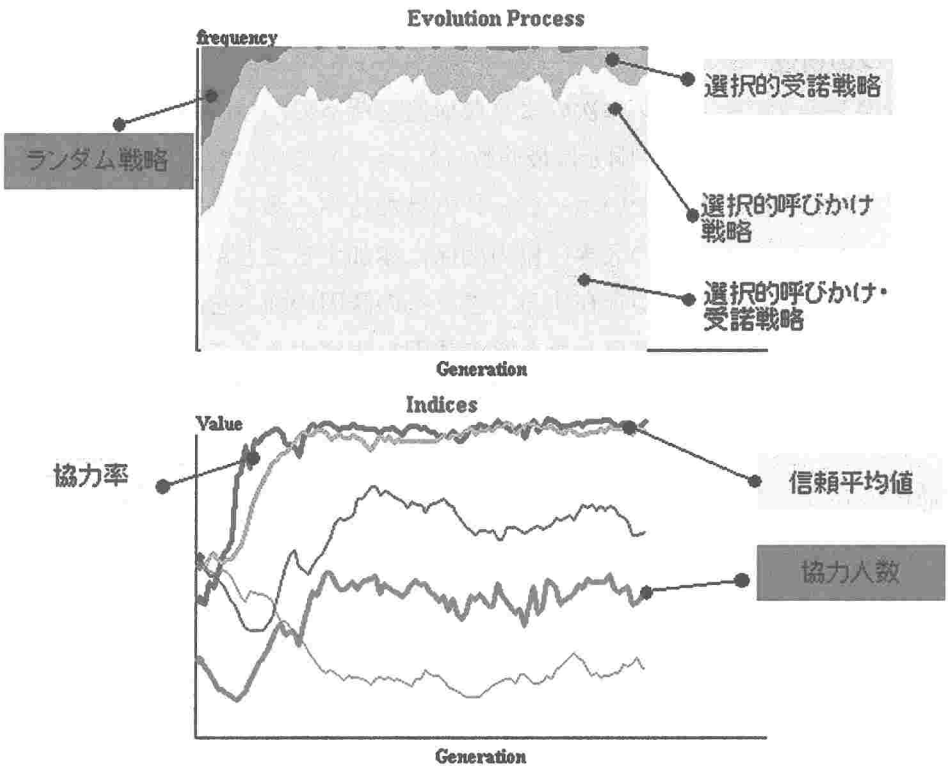


図1：シミュレーションの例
（賄賂条件、w=1.0）

図１の例は最も協力性が高まる条件の例である。他の条件、特に誘因が特大の条件では、後述するように、協力性は高まらず、協力人数も低い水準に留まるＲｕｎも観測された。

基礎指標

観測したエージェント社会の状況は次の３つの「基礎指標」で見ることができる。なお、２００世代を４０世代ごとに５つの世代ブロックに分け、世代ブロックを繰り返し要因として分散分析を実施した。

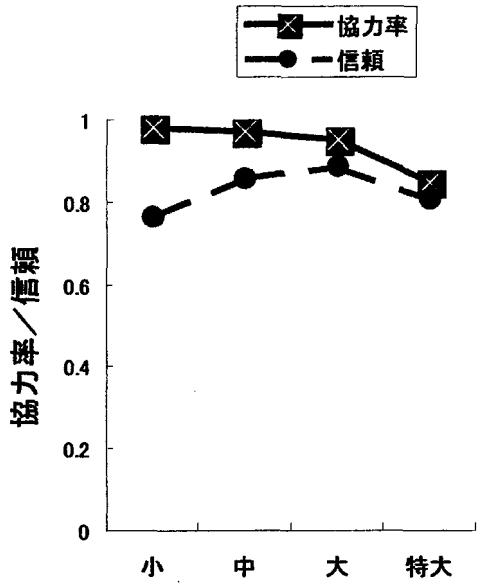


図2:協力率と信頼

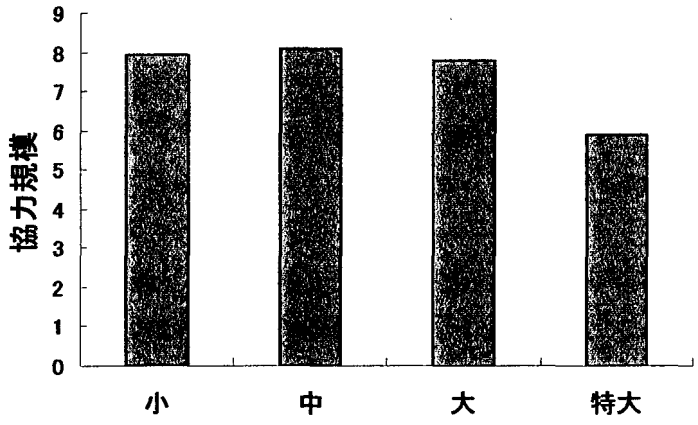


図3:協力関係への参加者数(協力規模)平均

まず協力率に対しては、違反誘因の強さの主効果だけが有意である（$F(3,72)=28.9$, $p<.000$）。図２にあるように違反誘因が高いほど協力率は低い。ただし協力率の差が出るのは「特大」とその他の水準との間であり、誘因が小中大の３水準間では有意な差はない(SNK 検定)。

協力への平均参加者数を示すのが図３である。同様に誘因の強さの主効果だけが有意な影響を及ぼす($F(3,72)=20.7, p<.000$)。差があるのは再び、特大とそれ以外の水準との間である（SNK)。誘因が小～大の範囲では、平均すれば最適規模より１人少ない数字の協力関係ができたことになる。

未知の相手に推論する協力率である信頼の平均値は、図２が示している。信頼に対しては、誘引の強さの主効果($F(3,72)=28.0, p<.000$)、および賄賂要因の主効果($F(1,72)=34.1, p<.000$)が有意となる。図２にその傾向が見られるように、信頼の平均は誘因中と大の間でだけ差がない（SNK)。そして協力率が有意に低かった特大条件を除けば、協力率に差がなかった誘因小～大の条件において、誘因が強い方が信頼性が高い、という結果が生じている。また、賄賂あり条件は賄賂なし条件の場合よりも、信頼は高くなっている。以上の結果は、誘因が特大であり従って協力自体が危うくなる条件を除けば、協力率は同じであっても、非協力を促す要因がある場合の方が信頼が高まっていることを示している。

戦略の変化

上記の結果は戦略分布の進化とも符合している。受入れと受諾が（ランダムではなく）選択的、つまり相手の違反率依存的である戦略を持つエージェントの比率に対しては、誘引の強さ要因の主効果（$F(3,72)=14.5, p<.000$)、および賄賂要因の主効果($F(1,72)=6.1, p<.02$)が有意となっている。選択的戦略は賄賂あり条件で高く、誘因小の条件だけが他の３水準より低くなっている(SNK)。すなわち、何れの要因によるにせよ、エージェントの非協力を促す条件が高いときに、戦略は選択的になる傾向が見て取れる。

考察

このシミュレーション結果の含意は次の２点にまとめることができる。

第１の含意は、選択的な inclusion（受入れ・受諾）の戦略がこのシミュレーションにおいても進化し、一定の協力性を安定的に確保できることがデモンストレイトされた。このとこは、選択的な inclusion に基づく「協力社会」の進化を予測することに論理的な矛盾が無いことを示している。

むろんこのシミュレーション結果がどの程度頑健であるかについては検討の余地がある。このシミュレーションでは、初期状態で戦略を無作為化したため、選択的な戦略が比較的多い状態から出発している。選択的戦略が初期状態で少ない条件で試行すれば、結果に変更が生じても不思議はない。

より印象的な結果は次の第２の含意として表現することができる。すなわち、違反誘因の強さによるにせよ賄賂オプションによるにせよ、社会の中で違反への

temptation が高い場合の方が（結果の協力水準では差が無いのに）、信頼（未知の相手に対して推定するデフォールトの協力確率）が高まる、という傾向があったことである。

この結果は次のように考えることができる。

temptation が低い場合は協力の発生を困難にする要因が当然ながら少ない。temptation が低い状態とは、山岸(1998, 1999)の表現を借りれば、エージェントが「針千本マシン」を飲んだような状態といえるだろう。つまり他のエージェントは放っておいても協力する傾向が生じる。このとき、戦略が選択的であることにはさして意味がない。要するに相手を（あまり）見ないでも済む（協力が生じると見込むことができる）。従って選択的な戦略の進化は、相対的に阻害されるだろう。

temptation が高い場合にはエージェントの戦略が選択的になることに意味がある。つまり、相手を見ながら非協力的な他者を関係に含めないようにするエージェントの方が生き残りやすい。シミュレーション結果は temptation が低い条件で戦略が選択的になり、それに伴って協力率が高まり、信頼も上がることを示した。ではなぜ、temptation が高いと信頼まで向上するのか？　次のように考えるべきだろう。このシミュレーションにおいても同様であるが、戦略が selective inclusion に基づくなら、信頼が高くない限り誰にも協力を呼びかけられず、結果として利得を失うことになる。だから、社会内の協力率がある程度達成されることを条件に、時折違反されるとしても、信頼を上げて協力関係に参与した方が、利益が高くなる（そのような戦略が生き残る）。だから temptation が高い条件で信頼も高くなるのだろう。

temptation による、信頼へのこのような効果は、inclusion という戦略状況が社会の中に built-in されてはじめて成り立つ。もし temptation の高い社会のエージェントと低い社会のエージェントを、同一の実験状況（同様の戦略分布が built-in されていない）に置いたら、trust の高い（temptation の高い社会の）エージェントの方がより協力的になって不思議はない。

同様に、temptation の高い社会のエージェントの方が、相手を見て判断することに慣れるため、相手の協力性を識別する能力を発達（進化）させても不思議はない。

引用文献

Foddy,M,, Smithson, M., Hogg, M. & Schneider, S. (1999) (Eds.) *Resolving Social Dilemmas.* NY: Psychology Press.

Kramer, R.M., McClintock, C.G. & Messick, D.M. (1986) Social values and cooperative response to a simulated resource conservation crisis. *Journal of Personality, 54,* 576-592.

Liebrand, W.B.G. & van Run, G.J. (1985) The effects of social motives on behavior in social dilemmas. *Journal of Experimental Social Psychology, 21,* 86-102.

山岸俊男（1998）『信頼の構造』、東京大学出版会.

山岸俊男（1999）『安心社会から信頼社会へ』、中央公論社（中公新書）.

資料
シミュレーションプログラムのソースコード

・本報告書で記述した研究で用いたシミュレーションプロ牛ラムのソースコードを以下に掲載する。
・各研究につき、最終的なプログラムとしていくつかのヴァージョンがある。以下では研究ごとに代表的なヴァージョン１つだけを掲載する。
・プログラムはすべて、Windows 版の Borland Delphi (Version.5 ないし 6)で書かれている。Delphi は、１つのプログラムを（ヴァージョンによって）６～８個のファイルで構成する。以下に記載するのはそのうちの Pascal ファイル（拡張子 .pas）だけである。

Ａ．権力創発モデル

```
{ Emergence of Power Project                    }
{ Punishment-Exchange Model. for recording No.1          }
{ Restricted Exchange Model                     }
{              Written in Borland Delphi Ver.5.0 J     }
{   Writing started on 2001.06.12               }
{  Eiji TAKAGI          2001.07.20              }
unit PunexR3a;

interface

uses
    Windows, Messages, SysUtils, Classes, Graphics,
Controls, Forms, Dialogs,
 ExtCtrls, StdCtrls, Buttons, Menus, jpeg;

type
 TForm1 = class(TForm)
   InfoBox: TPaintBox;
   MapBox: TPaintBox;
   ProcessBox: TPaintBox;
   EventBox: TPaintBox;
   StaticText1: TStaticText;
   StaticText2: TStaticText;
   Panel1: TPanel;
   Label1: TLabel;
   Label2: TLabel;
   Label3: TLabel;
   Label4: TLabel;
   Edit1: TEdit;
   Edit2: TEdit;
   Edit3: TEdit;
   Edit4: TEdit;
   Memo1: TMemo;
   RadioGroup1: TRadioGroup;
   RadioGroup2: TRadioGroup;
   RadioGroup3: TRadioGroup;
   Button1: TButton;
   BitBtn1: TBitBtn;
   BitBtn2: TBitBtn;
   Image1: TImage;
   CheckBox1: TCheckBox;
   CheckBox2: TCheckBox;
   CheckBox3: TCheckBox;
   CheckBox4: TCheckBox;
   procedure DefaultValues(Sender: TObject);
   procedure PaintB(Sender: TObject);
   procedure PaintB1(Sender: TObject);
   procedure InfoFrame(Sender: TObject);
   procedure StrategyMap(Sender: TObject);
   procedure DrawMap(Sender: TObject);
   procedure ProcessFrame(Sender: TObject);
   procedure EventFrame(Sender: TObject);
   procedure FormCreate(Sender: TObject);
   procedure BitBtn1Click(Sender: TObject);
   procedure BitBtn2Click(Sender: TObject);
{   procedure RadioGroup1Click(Sender: TObject);
   procedure RadioGroup2Click(Sender: TObject);}
   procedure Button1Click(Sender: TObject);
 private
   { Private 宣言 }
 public
   { Public 宣言 }
 end;

const
      MapSize    : array[1..2] of integer = (350, 350);
      ProcessSize : array[1..2] of integer = (400, 200);
      InfoBoxSize : array[1..2] of integer = (50, 50);
      EventSize   : array[1..2] of integer = (400, 200);
      ssize      = 15;
      NofP       = ssize*ssize; { N of players }
      NofSC      =  4; { N of Strength Classes }
      Pmutant    = 0.015; { Pr of Mutant Emergence }
      discount   = 1.00; { discount factor weight }
      v_f        = 2.0; { unit value of other's favor }
       PayColor    : array[0..NofSC-1] of TColor = (clYellow,
clLime, clSilver,
              clGray);{Payoff Class}
       SColor      : array[0..NofSC-1] of TColor = (clBlack,
clNavy, clBlue, clAqua);
       Pcolor      : array[0..NofSC-1] of TColor = (clYellow,
```

```
clMaroon, clRed, clFuchsia);
        Ccolor       : array[0..NofSC-1] of TColor = (clNavy,
clBlue, clFuchsia,
                clRed);
      Smark      : array[0..NofSC-1] of string[1]
              = ('0','1','2','3');
  type
     dig2      = array[1..2]  of byte;
     dig3      = array[1..3]  of byte;
     Person     = record
                   {MaxResource : byte;}
                   strength    : dig2;
                   Estrategy   : array[1..48] of byte;
                   Pstrategy   : array[1..16] of byte;
                   punish      : byte;
                   get         : byte;
                   res_level   : byte; { 1/2/3 }
                 end;
var
      Form1      : TForm1;
      Agent      : array[1..NofP] of Person;

implementation

{$R *.DFM}

{*****  function Strength Values  *****}
  function Vstrength(id : integer) : byte;
    begin
                      Vstrength  :=  Agent[id].strength[1]*2  +
Agent[id].strength[2];
    end; { of function Vstrength }

 {***  Converting digit vector into byte number  ***}
  function DtoB(x : dig3) : byte;
    var
        i, j, k, L     : integer;
        inc            : integer;
    begin
       j := 0; inc := 1;
       for i := 1 to 3 do begin
         if x[4-i] = 1 then j := j + inc;
         inc := inc *2;
       end; { of i loop }
       DtoB := j;
    end; { of DtoN }

{*****   Random Number Generator: Normal Distribution
*****}
  function Normal(m, s: real) : real;
{  inputs : m(mean) & s(standard deviation)          }
    var
        u, v    : array[1..2] of real;
        w, y    : real;
        i       : integer;
    begin
       w := 0.0;
       repeat
           for i := 1 to 2 do begin
             u[i] := random;
             v[i] := 2.0 * u[i] - 1.0;
           end; { of i loop }
           w := v[1]*v[1] + v[2]*v[2];
       until w <= 1.0;
       y := Sqrt(-2.0*Ln(w)/w);
       for i := 1 to 2 do
         u[i] := v[i]*y;
       Normal := u[1]*s + m;
    end; { of Normal }

procedure TForm1.DefaultValues(Sender: TObject);
    var
        i, j, k     : integer;
    begin
       Randomize;
       for i := 1 to NofP do begin
         if CheckBox3.checked then begin  {###}
           Agent[i].strength[1] := 1;
           Agent[i].strength[2] := 0;
         end
         else
         begin
           if random < 0.3 then begin
             Agent[i].strength[1] := 1;
             Agent[i].strength[2] := 1;
           end
           else
           begin
             Agent[i].strength[1] := 0;
             Agent[i].strength[2] := 1;
           end; {###}
           {for j := 1 to 2 do
            if random < 0.3 then Agent[i].strength[j] := 1
                      else Agent[i].strength[j] := 0;}
         end; { of if CheckBox3 }
         for j := 1 to 48 do
           Agent[i].Estrategy[j] := random(2);
         for j := 1 to 16 do
           Agent[i].Pstrategy[j] := random(2);
         Agent[i].get := 0;
         Agent[i].punish := 0;
       end; { of i Loop }
    end; { of DefaultValues }

procedure TForm1.StrategyMap(Sender: TObject);
    var
        i, j, k  : integer;
        InSt     : string[15];
    begin
       with MapBox.Canvas do begin
         Pen.Color := clBlack; Brush.Color:= clWhite;
         Rectangle(0, 0, Width, Height);
          Font.Color := clNavy; Font.Size := 12; Font.Style :=
[fsBold];
         TextOut(180, 3, 'Agent Map');
       end; { of with MapBox.Canvas }
    end; { of StrategyMap }

{*****    Drawing  Strategy  Map  &  Displaying  Strategy
Frequencies  *****}
  procedure TForm1.DrawMap(Sender: TObject);
    var
        i, j, k    : integer;
        InSt       : string[10];
        n1         : integer;

    procedure DrawPlayer(pL : integer);
      var
        i, j, k   : integer;
        Gx, Gy    : integer;
        Sid       : integer;
      begin
         i := pL mod ssize;
         if i = 0 then i := ssize;
         Gx := 25 + 20*(i-1);
         j := (pL-1) div ssize + 1;
         Gy := 25 + 20*(j-1);
          Sid := 3-Agent[pL].get;{3 - Agent[pL].Estrategy[1]*2 -
Agent[pL].Estrategy[2];}
         with MapBox.Canvas do begin
                          Pen.Color  :=  clNavy;  Font.Color  :=
Ccolor[Agent[pL].punish{Sid}]; Font.Size := 8;
                 Brush.Color  :=  PayColor[Sid];  Font.Style  :=
[fsBold];
           Rectangle(Gx, Gy, Gx+20, Gy+20);
           TextOut(Gx+5, Gy+2, Smark[Vstrength(pL)]);
         end; { of with MapBox.Canvas }
      end; { of DrawPlayer }

    begin
         for i := 1 to NofP do DrawPlayer(i);
      end; { of DrawMap }

procedure TForm1.InfoFrame(Sender: TObject);
```

```
    var
       i, j, k  : integer;
       InSt     : string[15];
       n1       : integer;
    begin
       with InfoBox.Canvas do begin
         Pen.Color := clBlack; Brush.Color:= clWhite;
         Rectangle(0, 0, Width, Height);
         {Font.Name := 'Arial';} Font.Style := [fsBold];
         Font.Color := clNavy; Font.Size := 11;
            TextOut(InfoBoxSize[1] div 2 -70, 2, '< Strategy
Distribution >');
         Font.Size := 10;
       end; { of with InfoBox.Canvas }
    end; { of InfoFrame }

  procedure TForm1.ProcessFrame(Sender: TObject);
   begin
       with ProcessBox.Canvas do begin
         Pen.Color := clBlack; Brush.Color:= clWhite;
         Rectangle(0, 0, Width, Height);
           {Font.Name := 'Times New Roman';} Font.Style :=
[fsBold];
         Font.Color := clNavy; Font.Size := 11;
         TextOut(ProcessSize[1] div 2 - 50, 2, 'Giving');
         Font.Color := clRed;
         TextOut(ProcessSize[1] div 2 + 50, 2, 'Punishing');
         Pen.Color := clBlack;
         MoveTo(10, 30); LineTo(10, ProcessSize[2]-20);
                          MoveTo(10,  ProcessSize[2]-20);
LineTo(ProcessSize[1]-10, ProcessSize[2]-20);
         Font.Color := clBlack; Font.Size := 10;
         TextOut(10,16, 'Level');
                        TextOut(ProcessSize[1]  div  2-30,
ProcessSize[2]-15, 'Generation');
       end; { of with ProcessBox.Canvas }
    end; { of ProcessFrame }

  procedure TForm1.EventFrame(Sender: TObject);
   begin
       with EventBox.Canvas do begin
         Pen.Color := clBlack; Brush.Color:= clWhite;
         Rectangle(0, 0, Width, Height);
           {Font.Name := 'Times New Roman';} Font.Style :=
[fsBold];
         Font.Color := clNavy; Font.Size := 11;
         TextOut(EventSize[1] div 2 -50, 2, 'Events');
         Pen.Color := clBlack;
         MoveTo(10, 30); LineTo(10, EventSize[2]-20);
                          MoveTo(10,  EventSize[2]-20);
LineTo(EventSize[1]-10, EventSize[2]-20);
         Font.Color := clBlack; Font.Size := 10;
         TextOut(10,16, 'Level');
            TextOut(EventSize[1] div 2-30, EventSize[2]-15,
'Generation');
       end; { of with EventBox.Canvas }
    end; { of EventFrame }

procedure TForm1.PaintB(Sender: TObject);
 begin
        InfoBox.Width := InfoBoxSize[1]; InfoBox.Height :=
InfoBoxSize[2];
     with InfoBox.Canvas do begin
       Brush.Color := clWindow;
       Rectangle(0, 0, Width, Height);
       Pen.Color := clWindow;
     end; { of with InfoBox }
       MapBox.Width  := MapSize[1];  MapBox.Height  :=
MapSize[2];
     with MapBox.Canvas do begin
       Brush.Color := clWindow;
       Rectangle(0, 0, Width, Height);
       Pen.Color := clWindow;
     end; { of with MapBox }
     ProcessBox.Width  := ProcessSize[1];
     ProcessBox.Height := ProcessSize[2];
     with ProcessBox.Canvas do begin
       Brush.Color := clWindow;
       Rectangle(0, 0, Width, Height);
       Pen.Color := clWindow;
     end; { of with ProcessBox }
     EventBox.Width  := EventSize[1];
     EventBox.Height := EventSize[2];
     with EventBox.Canvas do begin
       Brush.Color := clWindow;
       Rectangle(0, 0, Width, Height);
       Pen.Color := clWindow;
     end; { of with EventBox }
  end; { of PaintB }

procedure TForm1.PaintB1(Sender: TObject);
 begin
        InfoBox.Width := InfoBoxSize[1]; InfoBox.Height :=
InfoBoxSize[2];
     with InfoBox.Canvas do begin
       Brush.Color := clWindow;
       Rectangle(0, 0, Width, Height);
       Pen.Color := clWindow;
         {Font.Name := 'Times New Roman'; Font.Style :=
[fsBold];}
     end; { of with InfoBox }
       MapBox.Width  := MapSize[1];  MapBox.Height  :=
MapSize[2];
     with MapBox.Canvas do begin
       Brush.Color := clWindow;
       Rectangle(0, 0, Width, Height);
       Pen.Color := clWindow;
     end; { of with MapBox }
     ProcessBox.Width  := ProcessSize[1];
     ProcessBox.Height := ProcessSize[2];
     with ProcessBox.Canvas do begin
       Brush.Color := clWindow;
       Rectangle(0, 0, Width, Height);
       Pen.Color := clWindow;
     end; { of with ProcessBox }
     EventBox.Width  := EventSize[1];
     EventBox.Height := EventSize[2];
     with EventBox.Canvas do begin
       Brush.Color := clWindow;
       Rectangle(0, 0, Width, Height);
       Pen.Color := clWindow;
     end; { of with EventBox }
     DefaultValues(Sender);
     StrategyMap(Sender);
     ProcessFrame(Sender);
     InfoFrame(Sender);
     DrawMap(Sender);
     EventFrame(Sender);
  end; { of PaintB1 }

procedure TForm1.FormCreate(Sender: TObject);
begin
   PaintB1(Sender);
end;

procedure TForm1.Button1Click(Sender: TObject);
  var
       yn, gyn   : boolean;
       InSt      : string[10];
       Key_yn    : string[1];     { Definitions Y/N }
       Ccoef     : reaL;
       Icoef     : integer;
begin
      gyn := false;
      while not gyn do begin
  {      yn := false;
        while not yn do begin
          InSt := InputBox('Input', 'PsoLve[0.0-1.0]: ', '0.9');
          Ccoef := StrToFloat(InSt);
            if (Ccoef >= 0.0) and (Ccoef <= 1.0) then yn :=
true;
          if yn then PsoLve := Ccoef;
        end;} { of while yn }
        InSt := InputBox('Input', 'Above Definitions Correct ?
```

```
[y/n]: ', 'y');
        Key_yn := InSt;
        gyn := true;
        if (Key_yn = 'n') or (Key_yn = 'N') then gyn := false;
      end; { of while gyn }
end;

procedure TForm1.BitBtn2Click(Sender: TObject);
begin
   close;
end;

{Main Routine}
procedure TForm1.BitBtn1Click(Sender: TObject);

 const
     SSim      = 1;
     ESim      = 1;
     MaxResource = 24;
    { IPeriod   = 10}{200}{;}
 type
     posXY     = array[1..2] of byte;
     b_mat     = array[1..NofP,1..NofP] of byte;
     b_vec     = array[1..NofP] of byte;
     i_mat     = array[1..NofP,1..NofP] of integer;
     i_vec     = array[1..NofP] of integer;
     r_mat     = array[1..NofP,1..NofP] of reaL;
     r_vec     = array[1..NofP] of reaL;
     GrandRec  = record
                   give   : array[0..3] of reaL;
                   punish : array[0..3] of reaL;
                   gr_corr: reaL;
                 end;
     {r_mean    = array[1..2,0..3] of reaL;}
     AgentRec  = record
                   strength   : array[1..2] of byte;
                   Estrategy  : array[1..48] of byte;
                   Pstrategy  : array[1..16] of byte;
                   payoff     : reaL;
                 end;
 var
   {=== variables to record ===}
     give_id    : b_mat;
     punish_id  : b_mat;
     power_index : b_mat;
     Pay_Mat    : r_mat; { payoff flow matrix }
     Pay_Vec    : r_vec; { players' payoffs }
     give_rec   : b_mat; { give counts record for a round }
       g_r_corr     : array[0..1] of real; {correlation coeff. of
give-receive}
     puns_rec   : b_mat; { punish counts record for a round
}
     Mgive      : array[1..2, 0..NofSC-1] of reaL;
     Sgive      : array[0..NofSC-1] of reaL;
     Mpunish    : array[1..2, 0..NofSC-1] of reaL;
     Spunish    : array[0..NofSC-1] of reaL;
     Nstrength  : array[0..3] of integer;
   {=== variables for calc ===}
     res_rec    : array[1..5,1..NofP] of byte;
     rl_rec     : array[1..3,1..NofP] of byte;
     RClassid   : b_vec;

   {=== counter or identifier ===}
     converge   : boolean;   { identifier of run-convergence }
     iSim       : integer;   { counter of simulations }
     iRound     : integer;   { counter of rounds }
     do_repeat  : boolean;   { repeat the round or end }
          Ite       : integer;   { counter of iterations }
     ResC       : byte;      {Resource Size Coefficient}
   {=== variables first determined ===}
        NofRounds    : integer;  { Number of Rounds per
simulation }
     NofIte     : integer; { Number of Iterations per round }
     IPeriod    : integer; { Period of Initial strategy }
   {=== Condition Variables ===}
     DoMutant   : boolean; { Introduce Mutants }
   {=== File Specifications ===}
     FN_strategy  : string[50];
     Out_strategy : file of AgentRec;
     FN_mean      : string[50];
     Out_mean     : file of GrandRec;
     FN_give      : string[50];
     Out_give     : file of b_mat;{NNNNNN}
     FN_punish    : string[50];
     Out_punish   : file of b_mat;

{***** Drawing the Evolution Process per Iteration *****}
 procedure DrawProcess;
   const
      Ux = 3;
      {Uy = 15.0;}
   var
      i, j, k      : integer;
      ipage, roundId: integer;
      CumCnt       : array[1..2] of integer;
      Gxy          : array[1..2,1..2] of integer;
      xy           : array[1..4] of TPoint;
      Uy           : reaL;
   begin
      ipage := (iRound-1) div 100 + 1;
          roundId := iRound mod 100; if roundId = 0 then
roundId := 100;
      if roundId = 1 then begin
        with ProcessBox.Canvas do begin
          Pen.Color := clWhite; Brush.Color:= clWhite;
          Rectangle(0, 0, Width, Height);
          Pen.Color := clBlack;
          {Font.Name := 'Times New Roman';} Font.Style :=
[fsBold];
        Font.Color := clNavy; Font.Size := 11;
        TextOut(ProcessSize[1] div 2 - 50, 2, 'Giving');
        Font.Color := clRed;
        TextOut(ProcessSize[1] div 2 + 50, 2, 'Punishing');
        Pen.Color := clBlack;
        MoveTo(10, 30); LineTo(10, ProcessSize[2]-20);
                          MoveTo(10,  ProcessSize[2]-20);
LineTo(ProcessSize[1]-10, ProcessSize[2]-20);
        Font.Color := clBlack; Font.Size := 10;
        TextOut(10,16, 'Level');
                          TextOut(ProcessSize[1]  div  2-30,
ProcessSize[2]-15, 'Generation');
         { Pen.Color := clRed;
          MoveTo(Gf[3,1,1]-5, (Gf[3,1,2]+Gf[3,2,2]) div 2);
          LineTo(Gf[3,2,1], (Gf[3,1,2]+Gf[3,2,2]) div 2);}
        end; { of with ProcessBox.Canvas }
      end; { of if }
      {for i := 1 to 2 do CumCnt[i] := 0;}
      for i := 0 to NofSC-1 do begin
       { if (Mgive[1,i]>=0.0) or (Mgive[2,i]>=0.0) then begin}
        Uy := 45.0/Int(ResC);
        Gxy[1,1] := 10 + (roundId-1)*Ux;
        Gxy[2,1] := 10 + roundId*Ux;
        Gxy[1,2] := 180 - Round(Mgive[2,i]*Uy);
        Gxy[2,2] := 180 - Round(Mgive[1,i]*Uy);
        with ProcessBox.Canvas do begin
          Pen.Color := Scolor[i]; Brush.Color := Scolor[i];
          if iRound = 1 then
          begin

ellipse(Gxy[2,1]-1,Gxy[2,2]-1,Gxy[2,1]+1,Gxy[2,2]+1);
          end
          else
          begin
                                    MoveTo(Gxy[1,1],Gxy[1,2]);
LineTo(Gxy[2,1],Gxy[2,2]);
          end; { of if }
        end; { of with ProcessBox.Canvas }
        Uy := 15.0;
        Gxy[1,1] := 10 + (roundId-1)*Ux;
        Gxy[2,1] := 10 + roundId*Ux;
        Gxy[1,2] := 180 - Round(Mpunish[2,i]*Uy);
        Gxy[2,2] := 180 - Round(Mpunish[1,i]*Uy);
        with ProcessBox.Canvas do begin
          Pen.Color := Pcolor[i]; Brush.Color := Pcolor[i];
```

```
            if iRound = 1 then
            begin

ellipse(Gxy[2,1]-1,Gxy[2,2]-1,Gxy[2,1]+1,Gxy[2,2]+1);
            end
            else
            begin
                                    MoveTo(Gxy[1,1],Gxy[1,2]);
LineTo(Gxy[2,1],Gxy[2,2]);
            end; { of if }
          end; { of with ProcessBox.Canvas }

      end; { of i Loop }
   end; { of DrawProcess }

{***** Drawing the Evolution Process per Iteration *****}
  procedure DrawEvent;
    const
      Ux = 3;
      Uy = 80.0;
    var
      i, j, k      : integer;
      ipage, roundId: integer;
      CumCnt       : array[1..2] of integer;
      Gxy          : array[1..2,1..2] of integer;
      xy           : array[1..4] of TPoint;
    begin
      ipage  := (iRound-1) div 100 + 1;
          roundId := iRound mod 100; if roundId = 0 then
roundId := 100;
      if roundId = 1 then begin
        with EventBox.Canvas do begin
          Pen.Color := clWhite; Brush.Color:= clWhite;
          Rectangle(0, 0, Width, Height);
          Pen.Color := clBlack;
          {Font.Name := 'Times New Roman';} Font.Style :=
[fsBold];
        Font.Color := clNavy; Font.Size := 11;
        TextOut(EventSize[1] div 2 -50, 2, 'Events');
        Pen.Color := clBlack;
        MoveTo(10, 30); LineTo(10, EventSize[2]-20);
                                MoveTo(10,   EventSize[2]-20);
LineTo(EventSize[1]-10, EventSize[2]-20);
        Font.Color := clBlack; Font.Size := 10;
        TextOut(10,16, 'Level');
            TextOut(EventSize[1] div 2-30, EventSize[2]-15,
'Generation');
         { Pen.Color := clRed;
          MoveTo(Gf[3,1,1]-5, (Gf[3,1,2]+Gf[3,2,2]) div 2);
          LineTo(Gf[3,2,1], (Gf[3,1,2]+Gf[3,2,2]) div 2);}
        end; { of with EventBox.Canvas }
      end; { of if roundId = 1 }
      {for i := 1 to 2 do CumCnt[i] := 0;}
      {for i := 0 to NofSC-1 do begin}
       { if (Mgive[1,i]>=0.0) or (Mgive[2,i]>=0.0) then begin}
          Gxy[1,1] := 10 + (roundId-1)*Ux;
          Gxy[2,1] := 10 + roundId*Ux;
          Gxy[1,2] := 180 - Round((1.0+g_r_corr[0])*Uy);
          Gxy[2,2] := 180 - Round((1.0+g_r_corr[1])*Uy);
          with EventBox.Canvas do begin
            Pen.Color := cLNavy; Brush.Color := cLNavy;
            if iRound = 1 then
            begin

ellipse(Gxy[2,1]-1,Gxy[2,2]-1,Gxy[2,1]+1,Gxy[2,2]+1);
            end
            else
            begin
                                    MoveTo(Gxy[1,1],Gxy[1,2]);
LineTo(Gxy[2,1],Gxy[2,2]);
            end; { of if }
            {Polygon(xy);}
          end; { of with EventBox.Canvas }
      {end;} { of i Loop }
   end; { of DrawEvent }

{***** Global Initialization *****}
  procedure Global_Init;
   begin
      Randomize;
      IPeriod := 10;
      if CheckBox3.checked then IPeriod := 200;
   end; { of Global_Init }

{***** Simulation Initialization *****}
  procedure Sim_Init;

    procedure SetInitialValues;
     var
        i, j, k     : integer;
        iNofRounds  : integer;
        iResource   : integer;
        ResourceSize: byte;
     begin
        converge := false;
        iRound := 0;
        DoMutant := CheckBox1.Checked;
        iNofRounds := RadioGroup2.ItemIndex + 1;
        case iNofRounds of
         1 : NofRounds :=  100;
         2 : NofRounds :=  200;
         3 : NofRounds :=  300;
         4 : NofRounds :=  500;
        end; { of case iNofRounds }
        case RadioGroup1.ItemIndex of
         0 : NofIte := 100;
         1 : NofIte := 200;
        end; { of case RadioGroup1.ItemIndex }
        iResource := RadioGroup3.ItemIndex;
        case iResource of
         0 : ResourceSize :=  3;
         1 : ResourceSize :=  9;
         2 : ResourceSize := 12;
         3 : ResourceSize := 24;
        end; { of case iResource }
        ResC := ResourceSize div 3;
        for i := 1 to NofP do begin
          {Agent[i].MaxResource := ResourceSize;}
          if CheckBox3.checked then begin {###}
            Agent[i].strength[1] := 1;
            Agent[i].strength[2] := 0;
          end
          else
          begin
             if random < 0.3 then begin
               Agent[i].strength[1] := 1;
               Agent[i].strength[2] := 1;
             end
             else
             begin
               Agent[i].strength[1] := 0;
               Agent[i].strength[2] := 1;
             end; {###}
             {for j := 1 to 2 do
              if random < 0.3 then Agent[i].strength[j] := 1
                          else Agent[i].strength[j] := 0;}
          end; { of if CheckBox3 }
          for j := 1 to 48 do
            Agent[i].Estrategy[j] := random(2);
          {NNNNNN}
          { if random < 0.5 then
                    for j := 1 to 16 do Agent[i].Pstrategy[j] :=
random(2)
           else
             for j := 1 to 16 do
               if j mod 2 = 0 then Agent[i].Pstrategy[j] := 1
                          else Agent[i].Pstrategy[j] := 0;}
          for j := 1 to 16 do
            Agent[i].Pstrategy[j] := random(2);
          {NNNNNN}
          Agent[i].get := 0;
          Agent[i].punish := 0;
        end; { of i Loop }
        for i := 0 to 3 do Nstrength[i] := 0;
```

```
        for i := 1 to NofP do begin
          k := Agent[i].strength[1]*2 + Agent[i].strength[2];
          Nstrength[k] := Nstrength[k] + 1;
        end; { of i Loop }
        for i := 0 to 3 do
          for j := 1 to 2 do begin
            Mgive[j,i] := 0.0; Mpunish[j,i] := 0.0;
          end; { of j Loop }
    end; { of SetInitialValues }

  procedure OpenFiles;
    const
        PathName   = 'c:\PData3\';
    var
        i, j, k    : integer;
        FName      : string[50];
        Fno        : string[2];
    begin
        FName := '';
        {case RadioGroup3.ItemIndex of
         0 : FName := '03';
         1 : FName := '09';
         2 : FName := '12';
         3 : FName := '24';
        end;} { of case RadioGroup3.ItemIndex }
         if CheckBox2.checked then FName := Fname + 'P'
{with Punishment}
                          else FName := Fname + 'N'; {without
Punishment}
         if CheckBox4.checked then FName := Fname + 'S'
{Simple strategy}
                          else FName := Fname + 'C'; {Complex
strategy}
        Fno := IntToStr(iSim);
        if Length(Fno) < 2 then Fno := '0' + Fno;
        FN_strategy := PathName + Fname + Fno + 'A.dat';
        FN_mean     := PathName + Fname + Fno + 'B.dat';
        FN_give     := PathName + Fname + Fno + 'G.dat';
        if CheckBox2.checked then
          FN_punish   := PathName + Fname + Fno + 'P.dat';
                         AssignFile(Out_strategy,FN_strategy);
Rewrite(Out_strategy);
                              AssignFile(Out_mean,FN_mean);
Rewrite(Out_mean);
        AssignFile(Out_give,FN_give); Rewrite(Out_give);
        if CheckBox2.checked then begin
                            AssignFile(Out_punish,FN_punish);
Rewrite(Out_punish);
        end; { of if }
    end; { of OpenFiles }

  begin
      Edit4.Text := IntToStr(iSim);
      Edit4.Refresh;
      {DefaultValues(Sender);}
      SetInitialValues;
      OpenFiles;
      DrawMap(Sender);
  end; { of Sim_Init }

{*****  Round Initialization  *****}
  procedure Round_Init;
    var
      i, j    : integer;
      InSt    : string[10];
    begin
      iRound := iRound + 1;
      Edit1.Text := IntToStr(iRound);
      Edit1.Refresh;
      Edit3.Text := 'Running';
      Edit3.Refresh;
      Ite := 0;
      for i := 1 to NofP do begin
        Pay_Vec[i] := 0.0;
        Agent[i].punish := 0;
        for j := 1 to NofP do begin
          give_id[i,j] := 0;
          punish_id[i,j] := 0;
          power_index[i,j] := Vstrength(i)+1;
          give_rec[i,j] := 0;
          puns_rec[i,j] := 0;
        end; { of j loop }
        for j := 1 to 5 do
          res_rec[j,i] := 0;
        for j := 1 to 3 do
          rl_rec[j,i] := 0;
      end; { of i loop }
      for i := 0 to 3 do begin
        Sgive[i] := 0.0; Spunish[i] := 0.0;
      end; { of i Loop }
    end; { of Round_Init }

{*****  Repeat Initialization  *****}
  procedure Rep_Init;
    var
      i, j    : integer;
      InSt    : string[10];
    begin
      Ite := Ite + 1;
      Edit2.Text := IntToStr(Ite);
      Edit2.Refresh;
      for i := 1 to NofP do
        for j := 1 to NofP do begin
          Pay_Mat[i,j] := 0.0;
          {give_id[i,j] := 0;}
          punish_id[i,j] := 0;
        end; { of j Loop }
      {Edit3.Text := 'Rep_Init';
      Edit3.Refresh;}
    end; { of Rep_Init }

  procedure Get_power_index;
    var
      ppower, tpower   : real;
      pid, tid, xid    : integer;
      i, j, k          : integer;
        x0, y0     : byte;
        n, dist    : byte;
        ix, iy     : integer;
        jx, jy     : integer;
        dx, dy     : integer;
        Ncandidate: byte;
        candidate : array[1..24] of integer;

        TempId     : integer;
        {str50     : string[50];}
        pindex     : byte;

    begin
      for pid := 1 to NofP do begin
        x0 := pid mod ssize; y0 := (pid-1) div ssize + 1
        dist := 0; Ncandidate := 0;
        repeat
          dist := dist + 1;
          for i := 1 to 4 do begin
            case i of
              1 : begin
                    ix := x0 - dist; iy := y0;
                    dx := 1; dy := 1;
                  end;
              2 : begin
                    ix := x0; iy := y0 + dist;
                    dx := 1; dy := -1;
                  end;
              3 : begin
                    ix := x0 + dist; iy := y0;
                    dx := -1; dy := -1;
                  end;
              4 : begin
                    ix := x0; iy := y0 - dist;
                    dx := -1; dy := 1;
                  end;
            end; { of case i }
            for j := 1 to dist do begin
```

```
                jx := ix + (j-1) * dx; jy := iy + (j-1) * dy;
                if jx <  1 then jx := ssize+jx;
                if jx > ssize then jx := jx-ssize;
                if jy <  1 then jy := ssize+jy;
                if jy > ssize then jy := jy-ssize;
                TempId := jx + (jy-1)*ssize;
                Ncandidate := Ncandidate + 1;
                candidate[Ncandidate] := TempId;
              end; { of j Loop }
            end; { of i Loop }
        until dist = 3;
        for j := 1 to 24 do begin
          tid := candidate[j];
          ppower := 0.0; tpower := 0.0;
          for i := 1 to 24 do
          if i <> j then begin
            xid := candidate[i];
            if (xid <> pid) and (xid <> tid) then begin
              if (Pay_Mat[xid,pid] = 0.0) or (Pay_Mat[xid,tid] =
0.0) then begin
                pindex := Vstrength(xid);
                                        ppower := ppower +
Pay_Mat[xid,pid]*Int(pindex+1);
                                        tpower := tpower +
Pay_Mat[xid,tid]*Int(pindex+1);
              end; { of if }
            end; { of if }
          end; { of if / i(xid) Loop }
          pindex := Vstrength(pid);
          ppower := Int(pindex) + 1.0 + ppower/2.0;
          pindex := Vstrength(tid);
          tpower := Int(pindex) + 1.0 + tpower/2.0;
          power_index[pid,tid] := Round(ppower);
          {power_index[tid,pid] := Round(tpower);}
        end; { of j(tid) Loop }
      end; { of pid Loop }
    end; { of Get_power_index }

    function punishable(id, tid : integer) : boolean;
      begin
                if power_index[id,tid] > power_index[tid,id] +
1{power_index}{NNN}
          then punishable := true
          else punishable := false;
      end; { of function punishable }

{===  Calc of RClassid: resource levels of agents   ===}
  procedure Get_resource_level;
    var
      pid, tid        : integer;
      i, j            : integer;
      iseq            : byte;
      res_l, res_id   : array[1..NofP] of integer;
      ss              : real;
      max_l, temp_l   : integer;
      tempid          : integer;
      upperid, lowerid : byte;
      ulimit, llimit  : byte;
      done            : boolean;
      SClassid        : array[1..NofP] of byte;

    begin
      for pid := 1 to NofP do begin
        for i := 4 downto 1 do
          res_rec[i+1,pid] := res_rec[i,pid];
        ss := 0.0;
        for tid := 1 to NofP do
          if tid <> pid then ss := ss + Pay_Mat[tid,pid];
        res_rec[1,pid]    := Round(ss);
        res_id[pid] := pid;
      end; { of pid Loop }
      if Ite >= 5 then iseq := 5
              else iseq := Ite;
      for pid := 1 to NofP do begin
        ss := 0;
        for i := 1 to iseq do
          ss := ss + Int(res_rec[i,pid]);
      res_l[pid] := Round(ss/Int(iseq)) ;
    end; { of pid Loop }
  { arrange resorce level }
    for i := 1 to NofP-1 do begin
      max_l := res_l[i];
      for j := i+1 to NofP do
      if res_l[j] > max_l then begin
        temp_l := res_l[j]; res_l[j] := res_l[i];
        res_l[i] := temp_l;
        tempid := res_id[j]; res_id[j] := res_id[i]
        res_id[i] := tempid;
        max_l := res_l[i];
      end; { of if }
    end; { of i Loop }
  {  seek upper 1/4 }
    upperid := NofP div 4;
    i := upperid; done := false;
    repeat
        if res_l[i-1] > res_l[i] then
        begin
            done := true;
            ulimit := i;
        end
        else
        begin
            if i-1 = 1 then begin
              done := true;
              ulimit := 1;
            end; { of if i-1 }
        end; { of if }
        if not done then i := i - 1;
    until done;
    i := upperid; done := false;
    repeat
        if res_l[i+1] < res_l[i] then
        begin
            done := true;
            llimit := i;
        end
        else
        begin
            if i+1 = NofP then begin
              done := true;
              llimit := NofP;
            end; { of if i-1 }
        end; { of if }
        if not done then i := i + 1;
    until done;
    if llimit < NofP div 2 then upperid := llimit
      else
        if ulimit > 1 then upperid := ulimit
          else upperid := 0;
  {  seek lower 1/4 }
    lowerid := (NofP div 4) * 3;
    i := lowerid; done := false;
    repeat
        if res_l[i-1] > res_l[i] then
        begin
            done := true;
            ulimit := i;
        end
        else
        begin
            if i-1 = 1 then begin
              done := true;
              ulimit := 1;
            end; { of if i-1 }
        end; { of if }
        if not done then i := i - 1;
    until done;
    i := lowerid; done := false;
    repeat
        if res_l[i+1] < res_l[i] then
        begin
            done := true;
            llimit := i;
        end
```

```
            else
            begin
               if i+1 = NofP then begin
                  done := true;
                  llimit := NofP;
               end; { of if i-1 }
            end; { of if }
            if not done then i := i + 1;
        until done;
        if ulimit > NofP div 2 then lowerid := ulimit
          else
            if llimit > NofP then lowerid := llimit
              else lowerid := 0;
      {  Out results  }
       { str50 := IntToStr(upperid) + '-' + IntToStr(lowerid);
        form1.memo1.lines.add(Str50);}
        if (upperid = 0) and (lowerid = 0) then
          for i := 1 to NofP do SClassid[i] := 2;
        if (upperid = 0) and (lowerid <> 0) then
          for i := 1 to NofP do
            if i < lowerid then SClassid[i] := 2
                           else SClassid[i] := 3;
        if (upperid <> 0) and (lowerid = 0) then
          for i := 1 to NofP do
            if i <= upperid then SClassid[i] := 1
                            else SClassid[i] := 2;
        if (upperid <> 0) and (lowerid <> 0) then
          for i := 1 to NofP do
            if i <= upperid then SClassid[i] := 1  {res_id[i]}
              else if i < lowerid then  SClassid[i] := 2
                   else SClassid[i] := 3;
       { for i := 1 to NofP do begin
          Str50 := InttoStr(i) + ':' + intToStr(res_id[i]) + ' '
                                     +  IntToStr(res_l[i])  +  '='  +
IntToStr(SClassid[i]);
          form1.memo1.lines.add(Str50);
        end;} { of i Loop }
       { form1.memo1.lines.add('--------------------');}
        for i := 1 to NofP do
          RClassid[res_id[i]] := SClassid[i];
        for i := 1 to NofP do
          rl_rec[RClassid[i],i] := rl_rec[RClassid[i],i];
       {  for i := 1 to NofP do begin
          Str50 := InttoStr(i) + ':' + IntToStr(RClassid[i]);
          form1.memo1.lines.add(Str50);
        end;} { of i Loop }
    end; { of Get_resource_level }

{*****  Giving Phase  *****}
  procedure Giving_Phase;
    const
        Ugive     =  1.0;
        BudgetSize = 24.0;
    var
        i, j      : integer;
        str_mat    : b_mat;
        ss         : reaL;

    procedure AssignStrategy;
      var
        pid      : integer;
        n, dist  : byte;
        i, j, k  : integer;
        TempId   : integer;
        ix, iy   : integer;
        jx, jy   : integer;
        dx, dy   : integer;
        x0, y0   : byte;
        {str50    : string[50];}
        iPower, iPunish : byte;
      begin
        {Edit3.Text := 'Giving: Assign Strategy';
        Edit3.Refresh;}
         for pid := 1 to NofP do begin
          x0 := pid mod ssize; y0 := (pid-1) div ssize + 1;
          dist := 0; {Ncandidate := 0;}
          repeat
            dist := dist + 1;
            for i := 1 to 4 do begin
              case i of
                1 : begin
                      ix := x0 - dist; iy := y0;
                      dx := 1; dy := 1;
                    end;
                2 : begin
                      ix := x0; iy := y0 + dist;
                      dx := 1; dy := -1;
                    end;
                3 : begin
                      ix := x0 + dist; iy := y0;
                      dx := -1; dy := -1;
                    end;
                4 : begin
                      ix := x0; iy := y0 - dist;
                      dx := -1; dy := 1;
                    end;
              end; { of case i }
              for j := 1 to dist do begin
                jx := ix + (j-1) * dx; jy := iy + (j-1) * dy;
                if jx <  1 then jx := ssize+jx;
                if jx > ssize then jx := jx-ssize;
                if jy <  1 then jy := ssize+jy;
                if jy > ssize then jy := jy-ssize;
                TempId := jx + (jy-1)*ssize;
                iPower := 1;
                jx := power_index[pid,TempId];
                jy := power_index[TempId,pid];
                if jx > jy + 1 then iPower := 0;{power_index}
                if jy > jx + 1 then iPower := 2;{power_index}
                {iPunish := 0;}
                jx := 4*Rclassid[TempId];
                if Ite <= IPeriod then jx := 0;
                  iPunish := 2*Agent[TempId].Pstrategy[jx+3]
+ Agent[TempId].Pstrategy[jx+4];
                    if (Agent[TempId].Pstrategy[jx+3]=0) and
(Agent[TempId].Pstrategy[jx+4]=0)
                      then iPunish := 0;
                if give_id[TempId, pid] > 0 then iPunish := 0;
{NNNNN}
                {00 = 0, 01 = 1, 10 = 2, 11 := 3}
                 jx := 12*RClassid[pid] + 4*iPower + iPunish
+ 1;
                      if Ite <= IPeriod then jx := 4*iPower +
iPunish + 1;
                                          str_mat[pid,  TempId] :=
Agent[pid].Estrategy[jx];
              end; { of j Loop }
            end; { of i Loop }
          until dist = 3;
        end; { of pid Loop }
      end; { of AssignStrategy }

    procedure Exchange;
      var
        pid, tid  : integer;
        ss        : reaL;
      begin
        for pid := 1 to NofP do begin
          for tid := 1 to NofP do
            if pid <> tid then begin
              if str_mat[pid,tid] = 1 then
                Pay_Mat[pid,tid] := Ugive;
              if str_mat[pid,tid] = 2 then
                if (str_mat[tid,pid] = 1) or (str_mat[tid,pid] = 2)
then
                  Pay_Mat[pid,tid] := Ugive;
            end; { of if pid <> tid }
          ss := 0.0;
          for tid := 1 to NofP do
            if pid <> tid then ss := ss + Pay_Mat[pid,tid];
          Pay_Mat[pid,pid] := BudgetSize - ss;
        end; { of pid Loop }
      end; { of Exchange }
```

```
    begin
       {Edit3.Text := 'Giving';
       Edit3.Refresh;}
       for i := 1 to NofP do
         for j := 1 to NofP do
           str_mat[i,j] := 0;
       AssignStrategy;
       Exchange;
    end; { of Giving_Phase }

{*****  Punishing Phase  *****}
  procedure Punishing_Phase;
    var
       player     : integer; { player id }
       i          : integer;
       id1, id2   : byte;
       iPunish    : byte;
       Ncandidate : byte;
       candidate  : array[1..50] of integer;
       Ntarget    : integer;
        {str50    : string[50];}

     procedure Classify(pid : integer);
       var
          n, dist  : byte;
          i, j, k  : integer;
          TempId   : integer;
          ix, iy   : integer;
          jx, jy   : integer;
          dx, dy   : integer;
          x0, y0   : byte;
          iGive    : byte;
          DoPunish : boolean;
       begin
        {Edit3.Text := 'P-Classify';
        Edit3.Refresh;}
          x0 := pid mod ssize; y0 := (pid-1) div ssize + 1;
          dist := 0; Ncandidate := 0;
          repeat
              dist := dist + 1;
              for i := 1 to 4 do begin
                case i of
                  1 : begin
                        ix := x0 - dist; iy := y0;
                        dx := 1; dy := 1;
                      end;
                  2 : begin
                        ix := x0; iy := y0 + dist;
                        dx := 1; dy := -1;
                      end;
                  3 : begin
                        ix := x0 + dist; iy := y0;
                        dx := -1; dy := -1;
                      end;
                  4 : begin
                        ix := x0; iy := y0 - dist;
                        dx := -1; dy := 1;
                      end;
                end; { of case i }
                for j := 1 to dist do begin
                  jx := ix + (j-1) * dx; jy := iy + (j-1) * dy;
                  if jx <  1 then jx := ssize+jx;
                  if jx > ssize then jx := jx-ssize;
                  if jy <  1 then jy := ssize+jy;
                  if jy > ssize then jy := jy-ssize;
                  TempId := jx + (jy-1)*ssize;
                if punishable(pid,TempId) then {begin}
                    if Pay_Mat[pid,TempId] <= 0.0 then begin
{###}
                  iGive := 0; {not Given }
                    if Pay_Mat[TempId,pid] > 0.0 then iGive :=
1; {past Given }
                  k := 4*Rclassid[pid];
                  if Ite <= IPeriod then k := 0;
                  id1 := Agent[pid].Pstrategy[k+3];
                  id2 := Agent[pid].Pstrategy[k+4];
                          if (Agent[pid].Pstrategy[k+1]=0) and
(Agent[pid].Pstrategy[k+2]=0)
                          then begin
                            id1 := 0; id2 := 0;
                          end; { of if }
                          DoPunish := false;
                          if (id1 = 1) and (iGive = 1) then DoPunish :=
true;
                          if (id2 = 1) and (iGive = 0) then DoPunish :=
true;
                          if DoPunish then begin
                            Ncandidate := Ncandidate + 1;
                            candidate[Ncandidate] := TempId;
                          end; { of if DoPunish }
                        end; { of if Pay_Mat[pid,TempId] <= 0.0 }
                        end; { of j Loop }
                      end; { of i Loop }
              until dist = 3;
          end; { of Classify }

        procedure Punish(pid : integer);
          var
              i, n     : integer;
              str50    : string[50];
          begin
           {Edit3.Text := 'P-Punish';
           Edit3.Refresh;}
             Ntarget := {4}2*iPunish;
             if (Ncandidate > 0) and (Ntarget > 0) then begin
               n := 0;
               repeat
                   i := random(Ncandidate) + 1;
                   if candidate[i] > 0 then begin
                     punish_id[pid,candidate[i]]
                       := punish_id[pid,candidate[i]] + 2;
                     candidate[i] := 0;
                     n := n + 1;
                   end; { of if }
               until (n = Ncandidate) or (n = Ntarget);
             end; { of if Ncandidate > 0 }
          end; { of Punish }

        begin
           {Edit3.Text := 'Punishing';
           Edit3.Refresh;}
           {Get_power_index;}
           for player := 1 to NofP do begin
             i := 4*Rclassid[player];
             if Ite <= IPeriod then i := 0;
             id1 := Agent[player].Pstrategy[i+1];
             id2 := Agent[player].Pstrategy[i+2];
             iPunish := 2*id1 + id2;
             if iPunish > 0 then begin
               Classify(player);
               Punish(player);
             end; { of if iPunish > 0 }
           end; { of player Loop }
        end; { of Punishing_Phase }

{*****  Account  in each iteration  *****}
  procedure Account;
    const
      Ugive    = 1;
    var
      i, j, k  : integer;
      ss, tt   : reaL;
            PPay_Vec    : r_vec; { players' payoffs through
punishment }
      str50    : string[50];

     procedure Punishment(id : integer);
       var
           i, j, k  : integer;
           tid      : integer;
           ppower, tpower  : reaL;
           pn, tn          : integer;
           preward, treward : reaL;
       begin
```

```
        for tid := 1 to NofP do begin
          if tid <> id then
            if punish_id[id,tid] >= 2 then begin
              {ppower := 0.0; tpower := 0.0;} pn := 0; tn := 0;
              for i := 1 to NofP do
                if (i <> id) and (i <> tid) then begin
                  if (Pay_Mat[i,id] = 0.0) or (Pay_Mat[i,tid] =
0.0) then begin
                    {NNNNN}   {??? What's intent of the
prior version? }
                   if Pay_Mat[i, id] > 0.0 then pn := pn + 1;
                   if Pay_Mat[i,tid] > 0.0 then tn := tn + 1;
                   {NNNNN}
                  end; { of if }
                end; { of if }
              ppower := Int(power_index[id,tid]);
              tpower := Int(power_index[tid,id]);
              if ppower > tpower then begin
                preward := -1.0{5.0}{-0.5};
                treward := -5.0 - Abs(ppower-tpower);
              end; { of if }
              if ppower = tpower then begin
                preward := -2.0;
                treward := -2.0;
              end; { of if }
              if ppower < tpower then begin
                treward := -1.0{5.0}{-0.5};
                preward := -5.0 - Abs(ppower-tpower);
              end; { of if }
              if pn > 0 then
              begin
                    PPay_Vec[id] := PPay_Vec[id] +
preward/2.0;
                for i := 1 to NofP do
                  if (i <> id) and (i <> tid) then begin
                    if (Pay_Mat[i,id] = 0.0) or
(Pay_Mat[i,tid] = 0.0) then begin
                      if Pay_Mat[i,id] > 0.0 then
                        PPay_Vec[i] := PPay_Vec[i] +
preward/2.0/Int(pn);
                    end; { of if }
                  end; { of if }
              end
              else
              begin
                PPay_Vec[id] := PPay_Vec[id] + preward;
              end; { of if pn > 0 }
              if tn > 0 then
              begin
                    PPay_Vec[tid]:= PPay_Vec[tid] +
treward/2.0;
                for i := 1 to NofP do
                  if (i <> id) and (i <> tid) then begin
                    if (Pay_Mat[i,id] = 0.0) or
(Pay_Mat[i,tid] = 0.0) then begin
                      if Pay_Mat[i,tid] > 0.0 then
                        PPay_Vec[i] := PPay_Vec[i] +
treward/2.0/Int(tn);
                    end; { of if }
                  end; { of if }
              end
              else
              begin
                PPay_Vec[tid]:= PPay_Vec[tid] + treward;
              end; { of if tn > 0 }
            end; { of pinish_id }
        end; { of tid Loop }
      end; { of Punishment }

    begin
      {Edit3.Text := 'Account';
      Edit3.Refresh;}
      {for i := 1 to NofP do ExchangeClassify(i);}
        { Str50 := IntToStr(Ite) + ':';}
      for i := 1 to NofP do begin
        ss := 0.0;
        for j := 1 to NofP do
          if i <> j then ss := ss + Pay_Mat[i,j];
        Sgive[Vstrength(i)] := Sgive[Vstrength(i)] + ss;
        {if (i >= 100) and (i <= 104) then begin }
          {str50 := str50 + FloatToStrF(ss,ffFixed,10,1) + ' ';}
        {end;}
      end; { of i Loop }
        {form1.memo1.lines.add(str50);}
      if CheckBox2.checked then begin
      for i := 1 to NofP do begin
        ss := 0.0;
        for j := 1 to NofP do
          if i <> j then
            if punish_id[i,j] >= 2 then ss := ss + 1.0;
        Spunish[Vstrength(i)] := Spunish[Vstrength(i)] + ss;
      end; { of i Loop }

      for i := 1 to NofP do PPay_Vec[i] := 0.0;
      for i := 1 to NofP do Punishment(i);
      end; { of if CheckBox2.checked }
      for i := 1 to NofP do
        for j := 1 to NofP do begin
          if i <> j then begin
            give_id[i,j] := 0;
            if Pay_Mat[i,j] > 0.0 then give_id[i,j] := 1;
            Pay_Mat[i,j] := v_f*Pay_Mat[i,j];
          end; { of if i <> j Loop }
          if i = j then begin
            give_id[i,j] := Round(Pay_Mat[i,j]);
          end; { of if i <> j Loop }
        end; { of j Loop }
      for i := 1 to NofP do
        for j := 1 to NofP do {begin}
          if i <> j then begin   {NNNNNN}
            give_rec[i,j] := give_rec[i,j] + give_id[i,j];
            if punish_id[i,j] >= 2 then
              puns_rec[i,j] := puns_rec[i,j] + 1;
          end; { of if }
      tt := 1.0;
      if Ite > 1 then
        for i := 1 to (Ite-1) do
          tt := tt * discount;
      for i := 1 to NofP do begin
        if CheckBox2.checked then ss := PPay_Vec[i]
                             else ss := 0.0;
        for j := 1 to NofP do
          ss := ss + Pay_Mat[j,i];
        Pay_Vec[i] := Pay_Vec[i] + ss * tt;
      end; { of i loop }
    end; { of Account }

  {*****  Data Writing Out 1 *****}
    procedure DataOut1;
      var
        StrategyRec   : AgentRec;
        i, j, k       : integer;
      begin
        for i := 1 to NofP do begin
          for j := 1 to 2 do
            StrategyRec.strength[j] := Agent[i].strength[j];
          for j := 1 to 48 do
            StrategyRec.Estrategy[j] := Agent[i].Estrategy[j];
          for j := 1 to 16 do
            StrategyRec.Pstrategy[j] := Agent[i].Pstrategy[j];
          StrategyRec.payoff := Pay_Vec[i];
          Write(Out_strategy, StrategyRec);
        end; { of i Loop }
      end; { of DataOut1 }

  {*****   Round Calculation   *****}
    procedure RoundCalc;
      var
        i, j, k, L      : integer;
        N_rl            : array[1..3] of byte;
        Nchange         : array[0..3] of integer;{0: total, 1-3:
resource level}
        p_id            : i_vec;
        payment         : r_vec;
```

```
    max, min, dum   : reaL;
    imax, imin, idum : integer;
    done          : boolean;
     Uid            : array[0..3,1..30] of integer;{0: total, 1-3:
resource level}
    Lid           : array[1..30] of integer;
    tieid         : i_vec;
    ntie          : integer;
    Sid, Eid      : integer;
    scnt          : integer;
    str50         : string[50];
    f_vec         : array[0..3] of integer;

  procedure Get_rl_rec; { Resource Level of each Agent }
   var
     pid       : integer;
     i, j, k   : integer;
     max, maxid  : byte;
     candidate   : array[1..3] of byte;
     nn        : byte;
   begin
     for pid := 1 to NofP do begin
       maxid := 1; max := rl_rec[1,pid];
       for i := 2 to 3 do
         if rl_rec[i,pid] > max then begin
           maxid := i; max := rl_rec[i,pid];
         end; { of i Loop }
       nn := 0; for i := 1 to 3 do candidate[i] := 0;
       for i := 1 to 3 do
         if rl_rec[i,pid] = max then begin
           nn := nn + 1;
           candidate[nn] := i;
         end; { of if }
       k := random(nn) + 1;
       Agent[pid].res_level := candidate[k];
     end; { of pid Loop }
     for i := 1 to 3 do N_rl[i] := 0;
     for pid := 1 to NofP do
                                   N_rl[Agent[pid].res_level]   :=
N_rl[Agent[pid].res_level] + 1;
   end; { of Get_rl_rec }

  procedure Get_get_punish; { get-punish indices }
   var
     i, j, k    : integer;
   begin
     for i := 1 to NofP do begin
       if Pay_Vec[i] > 0.0 then
                                                 Agent[i].get   :=
Round(Pay_Vec[i]/(1800.0*Int(ResC))) + 1
        else Agent[i].get := 0;
       if Agent[i].get > 3 then Agent[i].get := 3;
       k := 0;
       for j := 1 to NofP do {###}
         if i <> j then
           k := k + puns_rec[i,j];
       k := k div (3*NofIte);
       if k >= 4 then k := 3;
       Agent[i].punish := k;
     end; { of i Loop }
   end; { of Get_get_punish }

   procedure Get_Superiors; { Getting Good Strategies for
each resource-class }
   var
     i, j, k, L  : integer;
     iclass     : byte;
   begin
     for iclass := 1 to 3 do
     if N_rl[iclass] > 0 then begin {###}
       Eid := 0; Sid := 1;
       for i := 1 to NofP do
         if Agent[i].res_level = iclass then begin
           Eid := Eid + 1;
           p_id[Eid] := i;
           payment[Eid] := Pay_Vec[i];
         end; { of if }
       Nchange[iclass] := Eid div 20{15} + 1;
     {+++  rearrange  +++}
       for i := Sid to Eid-1 do begin
         max := payment[i]; imax := p_id[i];
         for j := i to Eid do
           if payment[j] > max then begin
             dum := max; max := payment[j]; payment[j] :=
dum;
             payment[i] := max;
             idum := imax; imax := p_id[j]; p_id[j] := idum;
             p_id[i] := imax;
           end; { of if }
       end; { of i loop }
     {+++  seek upper  +++}
       max := payment[Sid+Nchange[iclass]-1];
       k := Sid + Nchange[iclass] - 1; done := false;
       repeat
         k := k - 1;
         if k = Sid - 1 then done := true;
         if k > Sid - 1 then
           if payment[k] > max then done := true;
       until done;
       k := k + 1;
       L := Sid + Nchange[iclass] - 1; done := false;
       repeat
         L := L + 1;
         if payment[L] < max then done := true;
         if L = Eid then done := true;
       until done;
       L := L - 1; if L = Eid then L := L + 1;
       ntie := L - k + 1;
       if k > Sid then
         for i := Sid to k-1 do Uid[iclass,i-Sid+1] := p_id[i];
       for i := k to L do tieid[i-k+1] := p_id[i];
       L := 0;
       repeat
         i := random(ntie) + 1;
         if tieid[i] <> 0 then begin
           L := L + 1;
           Uid[iclass,k-Sid+L] := tieid[i];
           tieid[i] := 0;
         end; { of if }
       until k-Sid+L = Nchange[iclass];
     end;  { of iclass Loop --> if N_rl[iclass] > 0 }
   end; { of Get_Superiors }

  procedure Get_Inferiors; { Getting Poor Strategies }
   var
     i, j, k, L  : integer;
   begin
     Eid := NofP; Sid := 1;
     for i := 1 to NofP do begin
       p_id[i] := i;
       payment[i] := Pay_Vec[i];
     end; { of i Loop }
     Nchange[0] := NofP div 20{15};
   {+++  rearrange  +++}
     for i := Sid to Eid-1 do begin
       max := payment[i]; imax := p_id[i];
       for j := i to Eid do
         if payment[j] > max then begin
           dum := max; max := payment[j]; payment[j] :=
dum;
           payment[i] := max;
           idum := imax; imax := p_id[j]; p_id[j] := idum;
           p_id[i] := imax;
         end; { of if }
     end; { of i loop }
   {+++  seek upper  +++}
     max := payment[Sid+Nchange[0]-1];
     k := Sid + Nchange[0] - 1; done := false;
     repeat
       k := k - 1;
       if k = Sid - 1 then done := true;
       if k > Sid - 1 then
         if payment[k] > max then done := true;
     until done;
```

```
      k := k + 1;
      L := Sid + Nchange[0] - 1; done := false;
      repeat
          L := L + 1;
          if payment[L] < max then done := true;
          if L = Eid then done := true;
      until done;
      L := L - 1; if L = Eid then L := L + 1;
      ntie := L - k + 1;
      if k > Sid then
        for i := Sid to k-1 do Uid[0,i-Sid+1] := p_id[i];
      for i := k to L do tieid[i-k+1] := p_id[i];
      L := 0;
      repeat
          i := random(ntie) + 1;
          if tieid[i] <> 0 then begin
            L := L + 1;
            Uid[0,k-Sid+L] := tieid[i];
            tieid[i] := 0;
          end; { of if }
      until k-Sid+L = Nchange[0];
    {+++  seek lower  +++}
      min := payment[Eid-Nchange[0]+1];
      k := Eid - Nchange[0] + 1; done := false;
      repeat
          k := k + 1;
          if k = Eid+1 then done := true;
          if k < Eid+1 then
            if payment[k] < min then done := true;
      until done;
      k := k - 1;
      L := Eid - Nchange[0] + 1; done := false;
      repeat
          L := L - 1;
          if payment[L] > min then done := true;
      until done;
      L := L + 1;
      ntie := k - L + 1;
      if k < Eid then
        for i := Eid downto k+1 do Lid[Eid-i+1] := p_id[i];
      for i := k downto L do tieid[k-i+1] := p_id[i];
      L := 0;
      repeat
          i := random(ntie) + 1;
          if tieid[i] <> 0 then begin
            L := L + 1;
            Lid[Eid-k+L] := tieid[i];
            tieid[i] := 0;
          end; { of if }
      until Eid-k+L = Nchange[0];
    end; { of Get_Inferiors }

procedure Shuffle;
  var
    i, j, k, L  : integer;
    iclass      : integer;
  begin
      for i := 1 to Nchange[0] do begin
        L := Lid[i];
       { k := random(Nchange[0]) + 1;
        k := Uid[0,k];
          for j := 1 to 48 do
            Agent[L].Estrategy[j] := Agent[k].Estrategy[j];
          for j := 1 to 16 do
            Agent[L].Pstrategy[j] := Agent[k].Pstrategy[j];}
        for iclass := 0 to 3 do begin
          k := random(Nchange[iclass]) + 1;
          k := Uid[iclass,k];
          for j := iclass*12+1 to iclass*12+12 do
            Agent[L].Estrategy[j] := Agent[k].Estrategy[j];
          for j := iclass*4+1 to  iclass*4+4 do
            Agent[L].Pstrategy[j] := Agent[k].Pstrategy[j];
        end; { of iclass }
      end; { of i loop }
    end; { of Shuffle }

begin
      {Edit3.Text := 'RoundCalc';
      Edit3.Refresh;}
      Get_rl_rec;
      Get_get_punish;
    {str50 := '';
    for iclass := 0 to 3 do begin
      f_vec[iclass] := 0;
      for i := 1 to NofP do begin
        j := Agent[i].Estrategy[1]*2 + Agent[i].Estrategy[2];
        f_vec[j] := f_vec[j] + 1;
      end;} { of i Loop }
      {str50 := str50 + IntToStr(f_vec[iclass]) + ':';
    end;} { of iclass Loop }
    {form1.memo1.lines.add(Str50);}
      Get_Superiors;
      Get_Inferiors;
      Shuffle;
    {+++  Strategy Frequencies Change  +++}
      for i := 0 to NofSC-1 do begin
        Mgive[2,i] := Mgive[1,i];
        if Nstrength[i] > 0 then
        Mgive[1,i] := Sgive[i]/Int(NofIte*Nstrength[i]);
        if CheckBox2.checked then begin
          Mpunish[2,i] := Mpunish[1,i];
          if Nstrength[i] > 0 then
                                                Mpunish[1,i]   :=
Spunish[i]/Int((NofIte-1)*Nstrength[i]);
        end; { of if CheckBox2.checked }
      end; { of i Loop }
  end; { of RoundCalc }

{*****  Producing Mutants  *****}
  procedure Mutant;
    var
      i, j, k     : integer;
      NewStrategy  : integer;
      scnt         : integer;
    begin
      if DoMutant then
      if not converge then
      for i := 1 to NofP do begin
        for j := 1 to 48 do
          if random < Pmutant then
                              if  Agent[i].Estrategy[j]  =  0  then
Agent[i].Estrategy[j] := 1
                                 else Agent[i].Estrategy[j] := 0;
            {case Agent[i].Estrategy[j] of
              0: if random < 0.5 then Agent[i].Estrategy[j] := 1
                                 else Agent[i].Estrategy[j] := 2;
              1: if random < 0.5 then Agent[i].Estrategy[j] := 0
                                 else Agent[i].Estrategy[j] := 2;
              2: if random < 0.5 then Agent[i].Estrategy[j] := 0
                                 else Agent[i].Estrategy[j] := 1;
            end;} { of case }

        for j := 1 to 16 do
          if random < Pmutant then
                              if  Agent[i].Pstrategy[j]  =  0  then
Agent[i].Pstrategy[j] := 1
                                 else Agent[i].Pstrategy[j] := 0;
      end; { of i Loop }
    {+++  Strategy Frequencies Adjustment  +++}
    {   for i := 0 to NofSC-1 do
          Sfreq[1,i] := 0;
        for i := 1 to NofP do begin
                          k  :=  3  -  Agent[i].Estrategy[1]*2  -
Agent[i].Estrategy[2];
          Sfreq[1,k] := Sfreq[1,k] + 1;
        end; }{ of i Loop }
    end; { of Mutant }

{*****  Judgnent if more repeated  *****}
  procedure RepeatJudgment;
    begin
      if iRound = NofRounds then converge := true;
    end; { of RepeatJudgment }
```

```
{***** Drawing the iteration Results *****}
 procedure DrawResults;

  procedure Disp_Round_Record;
   var
      i, j, k, tid : integer;
      str120       : string[120];
   begin
      for i := 1 to 3{NofP} do begin
        str120 := 'give_rec ' + IntToStr(i);
        form1.memo1.lines.add(str120);
        tid := 0;
        for j := 1 to 15 do begin
          str120 := '';
          for k := 1 to 15 do begin
            tid := tid + 1;
            str120 := str120 + IntToStr(give_rec[i,tid]) + ' ';
          end; { of k Loop }
          form1.memo1.lines.add(str120);
        end; { of j Loop }
      end; { of i Loop }
      for i := 1 to 3{NofP} do begin
        str120 := 'puns_rec ' + IntToStr(i);
        form1.memo1.lines.add(str120);
        tid := 0;
        for j := 1 to 15 do begin
          str120 := '';
          for k := 1 to 15 do begin
            tid := tid + 1;
            str120 := str120 + IntToStr(puns_rec[i,tid]) + ' ';
          end; { of k Loop }
          form1.memo1.lines.add(str120);
        end; { of j Loop }
      end; { of i Loop }
   end; { of Disp_Round_Record }

  procedure Disp_power_index;
   var
      i, j, k, tid : integer;
      str120       : string[120];
   begin
      for i := 1 to 3{NofP} do begin
        str120 := 'power_index ' + IntToStr(i);
        form1.memo1.lines.add(str120);
        tid := 0;
        for j := 1 to 15 do begin
          str120 := '';
          for k := 1 to 15 do begin
            tid := tid + 1;
            str120 := str120 + IntToStr(power_index[i,tid]) +
' ';

          end; { of k Loop }
          form1.memo1.lines.add(str120);
        end; { of j Loop }
      end; { of i Loop }
   end; { of Disp_power_index }

  procedure CalcCorr;
   var
      i, j, k, pid, tid : integer;
      mean, ss          : array[1..2] of reaL;
      variance          : array[1..2] of reaL;
      covar             : reaL;
      Str50             : string[50];
   begin
      g_r_corr[0] := g_r_corr[1];
      for i := 1 to 2 do begin
        mean[i] := 0.0;
        variance[i] := 0.0;
      end; { of i Loop }
      covar := 0.0;
      for i := 1 to NofP do  begin
        for j := 1 to 2 do ss[j] := 0.0;
        for j := 1 to NofP do
          if i <> j then begin
            ss[1] := ss[1] + Int(give_rec[i,j]);
            ss[2] := ss[2] + Int(give_rec[j,i]);
          end; { of if i <> j }
        for j := 1 to 2 do begin
          mean[j] := mean[j] + ss[j];
          variance[j] := variance[j] + ss[j]*ss[j];
        end; { of j Loop }
        covar := covar + ss[1]*ss[2];
      end; { of i Loop }
      for j := 1 to 2 do begin
        mean[j] := mean[j]/Int(NofP);
                                    variance[j]  :=  (variance[j]  -
Int(NofP)*mean[j]*mean[j])/Int(NofP-1);
      end; { of j Loop }
                                          covar  :=  (covar  -
Int(NofP)*mean[1]*mean[2])/Int(NofP-1);
      if (variance[1]>0.0) and (variance[2]>0.0) then
                                                g_r_corr[1]  :=
covar/(Sqrt(variance[1])*Sqrt(variance[2]))
      else
        g_r_corr[1] := 0.0;
      if iRound = NofRounds then begin
                  str50  :=  IntToStr(iRound)+  '  :  '  +
FloatToStrF(g_r_corr[1],ffFixed,10,3);
      form1.memo1.lines.add(str50);
      end; { of if }
    end; { of CalcCorr }

  begin
     DrawMap(Sender);
     DrawProcess;
     CalcCorr;
     {if CheckBox2.checked then}
       DrawEvent;
     {if (Run >= 49) then Disp_Round_Record;}
     {if (Run >= 49) then Disp_power_index;}
  end; { of DrawResults }

{***** Data Writing Out 2 *****}
 procedure DataOut2;
  var
    StrategyRec  : AgentRec;
    Gmean        : GrandRec;
    i, j, k      : integer;
  begin
     for i := 1 to NofP do begin
       for j := 1 to 2 do
         StrategyRec.strength[j] := Agent[i].strength[j];
       for j := 1 to 48 do
         StrategyRec.Estrategy[j] := Agent[i].Estrategy[j];
       for j := 1 to 16 do
         StrategyRec.Pstrategy[j] := Agent[i].Pstrategy[j];
       StrategyRec.payoff := Pay_Vec[i];
       Write(Out_strategy, StrategyRec);
     end; { of i Loop }
     Write(Out_give, give_rec);
     if CheckBox2.checked then
       Write(Out_punish, puns_rec);
     for i := 0 to NofSC-1 do begin
       Gmean.give[i] := Mgive[1,i];
       Gmean.punish[i] := Mpunish[1,i];
     end; { of i Loop }
     Gmean.gr_corr := g_r_corr[1];
     Write(Out_mean, Gmean);
  end; { of DataOut2 }

{*****  Results  *****}
 procedure Results;

  procedure Disp_Sim_Record;
   var
     i, j, k, L: integer;
     str50   : string[80];
   begin
     Edit3.Text := 'End';
     Edit3.Refresh;
     for i := 1 to NofP do begin
       str50 := '';
       { for j := 1 to 3 do
```

```
            str50 := str50 + IntToStr(Agent[i].Estrategy[j]);          end
          str50 := str50 + ' ';}
          L := 0;                                                      end
          for j := 1 to 4 do begin
            for k := 1 to 12 do begin
              L := L + 1;
              str50 := str50 + IntToStr(Agent[i].Estrategy[L]);
            end; { of k Loop }
            str50 := str50 + ' ';
          end; { of j Loop }
          form1.memo1.lines.add(str50);
          str50 := '';
          { for j := 1 to 2 do
            str50 := str50 + IntToStr(Agent[i].Pstrategy[j]);
          str50 := str50 + ' ';}
          L := 0;
          for j := 1 to 4 do begin
            for k := 1 to 4 do begin
              L := L + 1;
              str50 := str50 + IntToStr(Agent[i].Pstrategy[L]);
            end; { of k Loop }
            str50 := str50 + ' ';
          end; { of j Loop }
          str50 := str50 + ':' + IntToStr(i);
          str50 := str50 + ' ';
          { for j := 1 to 2 do
            str50 := str50 + IntToStr(Agent[i].strength[j]);}
                                    str50  :=  str50  +  '='  +
FloatToStrF(Pay_Vec[i],ffFixed,8,1);
          form1.memo1.lines.add(str50);
        end; { of i Loop }
      end; { of Disp_Sim_Record }

    procedure CloseFiles;
      begin
        CloseFile(Out_strategy); CloseFile(Out_mean);
        CloseFile(Out_give);
        if CheckBox2.checked then CloseFile(Out_punish);
      end; { of CloseFiles }

    begin
        DrawMap(Sender);
        {Disp_Sim_Record;}{NNNNNN}
        CloseFiles;
    end; { of Results }

{*****  Finale  *****}
  procedure Finale;
    begin
          form1.memo1.lines.add('Normal End of Job');
    end; { of Finale }

{=====  Main Routine  =====}
begin
    Global_Init;
    for iSim := SSim to ESim do begin
      Sim_Init;
      DataOut1;
      repeat
          Round_Init;
          repeat
              Rep_Init;
              Giving_Phase;
              get_resource_level;{NNN}
              get_power_index;
              if CheckBox2.checked then Punishing_Phase;
              Account;
          until Ite >= NofIte;
          RoundCalc;
          Mutant;
          RepeatJudgment;
          DrawResults;
          DataOut2;
      until converge;
      Results;
    end; { of iSim Loop }
    Finale;
```

B. Social Impact シミュレーションモデル

```
{ Adjustment Simulation program, N0.2-3              }
{              Written in Borland Delphi Ver.5.0 J   }
{  Eiji TAKAGI          2000.09.20              }
unit SI05a;

interface

uses
    Windows, Messages, SysUtils, Classes, Graphics,
Controls, Forms, Dialogs,
 ExtCtrls, StdCtrls, Buttons, Menus;

type
 TForm1 = class(TForm)
  MapBox: TPaintBox;
  ProcessBox: TPaintBox;
  BitBtn1: TBitBtn;
  BitBtn2: TBitBtn;
  Panel1: TPanel;
  Edit1: TEdit;
  Edit2: TEdit;
  Memo1: TMemo;
  RadioGroup1: TRadioGroup;
  RadioGroup2: TRadioGroup;
  RadioGroup3: TRadioGroup;
  RadioGroup4: TRadioGroup;
  RadioGroup5: TRadioGroup;
  CheckBox1: TCheckBox;
  Label2: TLabel;
  Label3: TLabel;
  StaticText1: TStaticText;
  StaticText2: TStaticText;
  CheckBox2: TCheckBox;
  Edit3: TEdit;
  Edit4: TEdit;
  Edit5: TEdit;
  Edit6: TEdit;
  Edit7: TEdit;
  Edit8: TEdit;
  Edit9: TEdit;
  Edit10: TEdit;
  Edit11: TEdit;
  Edit12: TEdit;
  CheckBox3: TCheckBox;
  CheckBox4: TCheckBox;
  CheckBox5: TCheckBox;
  MainMenu1: TMainMenu;
  help1: TMenuItem;
  procedure DefaultValues(Sender: TObject);
  procedure PaintB(Sender: TObject);
  procedure PaintB1(Sender: TObject);
  procedure StrategyMap(Sender: TObject);
  procedure DrawMap(Sender: TObject);
  procedure ProcessFrame(Sender: TObject);
     procedure DrawProcess(Sender: TObject; iRound :
integer);
  procedure FormCreate(Sender: TObject);
  procedure BitBtn1Click(Sender: TObject);
  procedure BitBtn2Click(Sender: TObject);
  procedure help1Click(Sender: TObject);

 private
  { Private 宣言 }
 public
  { Public 宣言 }
 end;

const
     SpaceSize  = 50;
     MapSize    : array[1..2] of integer = (400, 400);
     ProcessSize : array[1..2] of integer = (220, 350);
     CellColor  : array[1..9] of TColor
                  = (clWhite, clRed, clRed{Fuchsia},
clLime{Green}, clLime{Green},
                clBlue, clBlue{Aqua}, clBlack, clBlack{Gray});
     AColor : array[1..16] of TColor
         = (clFuchsia, clBlue, clRed, clYellow, clGreen,
           clWhite, clSilver, clLime, clAqua, clTeal, clOlive,
           clPurple, clMaroon, clNavy, clGray, clBlack);
     Color4 : array[1..4] of TColor
              = (clRed{Aqua}, clLime{Teal}, clYellow{Blue},
clBlue{Navy});
     NofA       = SpaceSize*SpaceSize; { N of Agents }
 type
    boolvec   = array[1..10] of boolean;
    Cell      = record
                  {dim       : boolvec;}
                  opinion    : byte;
                  {payoff    : reaL;}
                end;
var
     Form1      : TForm1;
      Agent, TAgent : array[1..SpaceSize, 1..SpaceSize] of
Cell;
     StdRed, StdGreen, StdBlack, StdBlue : boolvec;
     DimSize    : byte;
     StdMax     : array[1..2] of byte;
     NofG       : byte;
     Gfreq      : array[1..2,1..16] of integer;
      { group frequencies recorder }
      {1..2 =  1: current Round, 2: previous round }
     N_change   : array[1..5] of integer;
     MaxDist    : byte;
     MaxDist1   : reaL;
      Asize     : array[1..5] of integer; {Agent Group Size for
each opinion }

implementation

{$R *.DFM}

{=================================}
 function AgentColor(ix : byte): Tcolor;
  begin
     AgentColor := AColor[ix];
  end; { of function AgentColor }

{##########  Initial Values  ############}
 procedure TForm1.DefaultValues(Sender: TObject);
    var
       i, j, k, L     : integer;
       InitialDist    : reaL;
       str10          : string[10];
    begin
       randomize;
       DimSize := RadioGroup3.ItemIndex + 2;
       {case RadioGroup3.ItemIndex of
        0 : begin
              DimSize := 2;
              StdMax[1] := 2; StdMax[2] := 1;
            end;
        1 : begin
              DimSize := 2;
              StdMax[1] := 2; StdMax[2] := 1;
            end;
        2 : begin
              DimSize := 4;
              StdMax[1] := 3; StdMax[2] := 4;
            end;
      end;} { of case RadioGroup3 }
      case RadioGroup4.ItemIndex of
       0 : InitialDist := 0.4;
       1 : InitialDist := 0.3;
       2 : InitialDist := 0.2;
       3 : InitialDist := 0.1;
      end; { of case RadioGroup4 }
      MaxDist :=  15;{CCC}
```

```
      MaxDist1 := 10.0;
      if CheckBox2.Checked then begin
        MaxDist :=  2*MaxDist;
        MaxDist1 := 2.0*MaxDist1;
      end; { of if }
      for i := 1 to SpaceSize do
        for j := 1 to SpaceSize do
          if random < InitialDist then Agent[i,j].opinion := 1
          else Agent[i,j].opinion := random(DimSize-1)+2;
          {Agent[i,j].opinion := random(DimSize) + 1;}
          {Agent[i,j].payoff := 0.0;}
        {end;} { of j Loop }
         { k := random(10)+1;
          case k of
           1..7 : begin
                    Agent[i,j].opinion := 1;
                  end;
           8..10 : begin
                    Agent[i,j].opinion := 2;
                  end;
           10..12: begin
                    Agent[i,j].opinion := 3;
                  end;
           10  : begin
                    Agent[i,j].opinion := 4;
                  end;
           10: begin
                    Agent[i,j].opinion := 5;
                  end;
          end;} { of case k }
          { k := random(2);
          if j <= 25 then begin
            if random < 0.8 then k := 0 else k := 1;
          end
          else
          begin
            if random < 0.8 then k := 1 else k := 0;
          end;
          case k of
            0   : for L := 1 to 10 do Agent[i,j].dim[L] := true;
            1   : for L := 1 to 10 do Agent[i,j].dim[L] := false;
          end;} { of case k }
        {end; { of j Loop }
      for i := 1 to DimSize do Asize[i] := 0;
      for i := 1 to SpaceSize do
        for j := 1 to SpaceSize do
                                  Asize[Agent[i,j].opinion]   :=
Asize[Agent[i,j].opinion] + 1;
      for i := 1 to DimSize do begin
        str10 := IntToStr(Asize[i]);
        case i of
          1 : begin
                Edit3.Text := str10;
                Edit3.Refresh;
              end;
          2 : begin
                Edit4.Text := str10;
                Edit4.Refresh;
              end;
          3 : begin
                Edit5.Text := str10;
                Edit5.Refresh;
              end;
          4 : begin
                Edit6.Text := str10;
                Edit6.Refresh;
              end;
          5 : begin
                Edit7.Text := str10;
                Edit7.Refresh;
              end;
        end; { of case i }
      end; { of for i Loop }
      NofG := DimSize;
      {for i := 1 to DimSize do NofG := NofG * 2;
      for i := 1 to DimSize do begin
        StdRed[i] := true; StdGreen[i] := false;
        end;} { of i Loop }
        for i := 1 to DimSize div 2 do begin
          StdBlack[i] := true; StdBlue[i] := false;
        end; { of i Loop }
        for i := DimSize div 2 + 1 to DimSize do begin
          StdBlack[i] := false; StdBlue[i] := true;
        end; { of i Loop }
        for i := 1 to 5 do N_change[i] := 0;
      end; { of DefaultValues }

{###########  Landscape Map  ############}
 procedure TForm1.StrategyMap(Sender: TObject);
    var
       i, j, k  : integer;
       InSt     : string[15];
    begin
       with MapBox.Canvas do begin
         Pen.Color := clBlack; Brush.Color:= clBlack;
         Rectangle(0, 0, Width, Height);
       end; { of with MapBox.Canvas }
    end; { of StrategyMap }

{###########  Drawing Landscape Map  ###########}
  procedure TForm1.DrawMap(Sender: TObject);
    var
       i, j  : byte;

    procedure DrawPlayer(ix,iy : byte);
      var
        i, j, k   : integer;
        Gx, Gy    : integer;
        Sid, Aid  : integer;
      begin
        Gx := 8*(ix-1); Gy := 8*(iy-1);
        with MapBox.Canvas do begin
                                          Pen.Color          :=
clBlack{AgentColor(Agent[ix,iy].dim)};
          Brush.Color := AgentColor(Agent[ix,iy].opinion);
          Rectangle(Gx, Gy, Gx+8, Gy+8);
        end; { of with MapBox.Canvas }
      end; { of DrawPlayer }

    begin
       for i := 1 to SpaceSize do
         for j := 1 to SpaceSize do DrawPlayer(i,j);
    end; { of DrawMap }

   procedure TForm1.ProcessFrame(Sender: TObject);
     begin
       with ProcessBox.Canvas do begin
         Pen.Color := clBlack; Brush.Color:= clWhite;
         Rectangle(0, 0, Width, Height);
           Font.Name := 'Times New Roman'; Font.Style :=
[fsBold];
         Font.Color := clNavy; Font.Size := 11;
           TextOut(ProcessSize[1] div 2 -50, 2, 'Evolution
Process');
         Pen.Color := clBlack;
         MoveTo(10, 30); LineTo(10, ProcessSize[2]-20);
                          MoveTo(10,   ProcessSize[2]-20);
LineTo(ProcessSize[1]-10, ProcessSize[2]-20);
         Font.Color := clBlack; Font.Size := 10;
         TextOut(10,16, 'frequency');
                          TextOut(ProcessSize[1]  div  2-30,
ProcessSize[2]-15, 'Round');
       end; { of with ProcessBox.Canvas }
     end; { of ProcessFrame }

{***** Drawing the Evolution Process per Iteration *****}
  procedure TForm1.DrawProcess(Sender: TObject; iRound
 : integer);
    const
       Ux =  4;
       Uy = 10;
    var
       i, j, k     : byte;
       ipage, jround : integer;
```

```
      CumCnt      : array[1..2] of integer;
      Gxy         : array[1..2,1..2] of integer;
      xy          : array[1..4] of TPoint;
      NofP        : integer;
  begin
      NofP := SpaceSize*SpaceSize;
      ipage  := (iRound-1) div 100 + 1;
      jround := iRound mod 100; if jround = 0 then jround :=
100;
      if jround = 1 then begin
        with ProcessBox.Canvas do begin
          Pen.Color := clWhite; Brush.Color:= clWhite;
                      Rectangle(10,  30,  ProcessSize[1]-10,
ProcessSize[2]-20);
          Pen.Color := clBlack;
          MoveTo(10, 30); LineTo(10, ProcessSize[2]-20);
          MoveTo(10, ProcessSize[2]-20);
          LineTo(ProcessSize[1]-10, ProcessSize[2]-20);
        end; { of with ProcessBox.Canvas }
      end; { of if }
      for i := 1 to 2 do CumCnt[i] := 0;
      for i := NofG downto 1 do
        if (Gfreq[1,i]>0) or (Gfreq[2,i]>0) then begin
          Gxy[1,1] := 10 + (jround-1)*Ux;
          Gxy[2,1] := 10 + jround*Ux;
              Gxy[1,2] := ProcessSize[2]-20-NofP div Uy +
CumCnt[2] div Uy;
              Gxy[2,2] := ProcessSize[2]-20-NofP div Uy +
CumCnt[1] div Uy;
          xy[1] := Point(Gxy[1,1], Gxy[1,2]);
          xy[2] := Point(Gxy[2,1], Gxy[2,2]);
              for j := 1 to 2 do CumCnt[j] := CumCnt[j] +
Gfreq[j,i];
              Gxy[1,2] := ProcessSize[2]-20-NofP div Uy +
CumCnt[2] div Uy;
              Gxy[2,2] := ProcessSize[2]-20-NofP div Uy +
CumCnt[1] div Uy;
          xy[4] := Point(Gxy[1,1], Gxy[1,2]);
          xy[3] := Point(Gxy[2,1], Gxy[2,2]);
          with ProcessBox.Canvas do begin
              Pen.Color := AgentColor(i); Brush.Color :=
AgentColor(i);
            Polygon(xy);
          end; { of with ProcessBox.Canvas }
        end; { of if }
  end; { of DrawProcess }

{###########  Form Paint  ############}
procedure TForm1.PaintB(Sender: TObject);
 begin
      MapBox.Width  := MapSize[1];  MapBox.Height  :=
MapSize[2];
    with MapBox.Canvas do begin
      Brush.Color := clWindow;
      Rectangle(0, 0, Width, Height);
      Pen.Color := clWindow;
    end; { of with MapBox }
 end; { of PaintB }

{###########  Form Paint 1  ############}
procedure TForm1.PaintB1(Sender: TObject);
 begin
      MapBox.Width  := MapSize[1];  MapBox.Height  :=
MapSize[2];
    with MapBox.Canvas do begin
      Brush.Color := clBlack;
      Rectangle(0, 0, Width, Height);
      Pen.Color := clWindow;
    end; { of with MapBox }
    DefaultValues(Sender);
    StrategyMap(Sender);
    DrawMap(Sender);
    ProcessFrame(Sender);
 end; { of PaintB1 }

{###########  Form Create  ############}
procedure TForm1.FormCreate(Sender: TObject);
begin
   PaintB1(Sender);
end;

{###########  Close Button  ###########}
procedure TForm1.BitBtn2Click(Sender: TObject);
begin
   close;
end;

{###########  Main Routine  ###########}
procedure TForm1.BitBtn1Click(Sender: TObject);

 var
    iSim      : integer;   { counter of Sim iteration }
    iRound    : integer;   { counter of round }
    NofRounds : integer;
    iAgent    : integer;
    iAx, iAy  : byte;
    A_id      : integer;
    P_try     : reaL;
    NChange   : integer;
    end_round : boolean;
    TurnOrder : array[1..NofA] of integer;
    Entropy   : reaL;

{ procedure GetIndex(var entrp : reaL);
  var
    Ientropy  : array[0..1023] of integer;
    i, j, k   : integer;
    n, bit,num  : integer;
    x, y      : reaL;
  begin
    for i := 0 to 1023 do Ientropy[i] := 0;
    for i := 1 to SpaceSize do
      for j := 1 to SpaceSize do begin
        n := 1; num := 0;
        for k := DimSize downto 1 do begin
          if Agent[i,j].dim[k] then bit := 1 else bit := 0;
          num := num + bit*n;
          n := n*2;
        end;} { of k Loop }
      {   Ientropy[num] := Ientropy[num] + 1;
      end;} { of j Loop }
    { y := 0.0;
    for i := 0 to 1023 do
      if Ientropy[i] > 0 then begin
        x := Int(Ientropy[i])/Int(NofA);
        y := y - x * Ln(x);
      end;} { of i Loop }
    { entrp := y;
  end; }{ of GetIndex }

{*****  Global Initialization  *****}
 procedure GLobaLInit;
  begin
    Randomize;
    case RadioGroup1.ItemIndex of
      0 : NofRounds :=  50;
      1 : NofRounds := 100;
      2 : NofRounds := 150;
      3 : NofRounds := 500;
    end; { of case RadioGroup1 }
    case RadioGroup2.ItemIndex of
      0 : P_try := 0.1;
      1 : P_try := 0.2;
      2 : P_try := 0.5;
    end; { of case RadioGroup1 }
  end; { of GLobaLInit }

{*****  Simulation Run Initialization  *****}
 procedure SimInit;
  var
     i, j, k, L  : integer;
     str50   : string[50];

  procedure EditClear;
```

```
    var
      str10  : string[10];
    begin
      str10 := '';
      Edit3.Text := str10;
      Edit3.Refresh;
      Edit4.Text := str10;
      Edit4.Refresh;
      Edit5.Text := str10;
      Edit5.Refresh;
      Edit6.Text := str10;
      Edit6.Refresh;
      Edit7.Text := str10;
      Edit7.Refresh;
      Edit8.Text := str10;
      Edit8.Refresh;
      Edit9.Text := str10;
      Edit9.Refresh;
      Edit10.Text := str10;
      Edit10.Refresh;
      Edit11.Text := str10;
      Edit11.Refresh;
      Edit12.Text := str10;
      Edit12.Refresh;
    end; { of EditClear }

  begin
    EditClear;
    DefaultValues(Sender);
    StrategyMap(Sender);
    DrawMap(Sender);
    ProcessFrame(Sender);
    end_round := false;
    iRound := 0;
    { GetIndex(Entropy);  }
    str50 := IntToStr(iRound) + ' : ';
    { str50 := str50 + FloatToStrF(Entropy,ffFixed,10,4);}
    form1.memo1.lines.add(str50);
    for i := 1 to NofG do Gfreq[1,i] := 0;
    for i := 1 to SpaceSize do
      for j := 1 to SpaceSize do
                                  Gfreq[1,Agent[i,j].opinion]   :=
Gfreq[1,Agent[i,j].opinion] + 1;
  end; { of SimInit }

{*****  Round Initialization  *****}
 procedure RoundInit;
  var
    i, j  : integer;

  procedure PutTurnOrder;
   var
    i, j, k   : integer;
    seL1, seL2 : integer;
   begin
     for i := 1 to NofA do TurnOrder[i] := i;
     for i := 1 to 10 do
       for j := 1 to NofA do begin
                        seL1  :=  random(NofA)+1;  seL2  :=
random(NofA)+1;
         k := TurnOrder[seL1];
         TurnOrder[seL1] := TurnOrder[seL2];
         TurnOrder[seL2] := k;
       end; { of j Loop }
   end; { of PutTurnOrder }

  begin
    iRound := iRound + 1;
    Edit1.Text := IntToStr(iRound);
    Edit1.Refresh;
    PutTurnOrder;
    NChange := 0;
    if not CheckBox5.Checked then
    for i := 1 to SpaceSize do
      for j := 1 to SpaceSize do
        TAgent[i,j].opinion := Agent[i,j].opinion;
  end; { of RoundInit }

{*****  Loop Initialization  *****}
 procedure LoopInit;
  begin
    Edit2.Text := IntToStr(A_id);
    Edit2.Refresh;
    iAx := iAgent mod SpaceSize;
    if iAx = 0 then iAx := SpaceSize;
    iAy := (iAgent-1) div SpaceSize + 1;
  end; { of LoopInit }

{*****  Agent's Trial to Change  *****}
 procedure TryChange;
  var
    i, j    : integer;
    Pressure : array[1..10] of reaL{byte};
    maxval   : reaL;
    maxid    : array[1..10] of byte;
    ntie     : byte;
    NewOpinion : byte;

   procedure CalcPressure(ix, iy : byte);
    var
      i, j, k   : integer;
      n, dist   : byte;
      tx, ty    : integer;
      ux, uy    : integer;
      jx, jy    : integer;
      dx, dy    : integer;
      rd, rd1   : reaL;
      agr_value : reaL;
    begin
      if RadioGroup5.ItemIndex = 0 then MaxDist := 1;
      for i := 1 to DimSize do Asize[i] := 0;
      if RadioGroup5.ItemIndex <= 1 then rd1 := 1.0;
      rd := 0.84;
      if RadioGroup5.ItemIndex > 1 then
        case RadioGroup5.ItemIndex of
          2 : rd1 := rd;
          3 : rd1 := rd * rd;
          4 : rd1 := rd * rd * rd;
          5 : rd1 := rd * rd * rd * rd;
        end; { of case }
      if not CheckBox3.Checked then begin
        if not CheckBox5.Checked then
        begin
        Pressure[TAgent[iAx,iAy].opinion]
          := Pressure[TAgent[iAx,iAy].opinion] + 1.0/rd1;
                              Asize[TAgent[iAx,iAy].opinion]  :=
Asize[TAgent[iAx,iAy].opinion] + 1;
        end
        else
        begin
        Pressure[Agent[iAx,iAy].opinion]
          := Pressure[Agent[iAx,iAy].opinion] + 1.0/rd1;
                                Asize[Agent[iAx,iAy].opinion]  :=
Asize[Agent[iAx,iAy].opinion] + 1;
        end; { of if not CheckBox5.Checked }
      end; { of if not CheckBox3.Checked }
      dist := 0;
      repeat
        dist := dist + 1;
        for i := 1 to 4 do begin
          case i of
            1 : begin
                  tx := ix - dist; ty := iy;
                  dx := 1; dy := 1;
                end;
            2 : begin
                  tx := ix; ty := iy + dist;
                  dx := 1; dy := -1;
                end;
            3 : begin
                  tx := ix + dist; ty := iy;
                  dx := -1; dy := -1;
                end;
            4 : begin
```

```
            tx := ix; ty := iy - dist;
            dx := -1; dy := 1;
          end;
        end; { of case i }
        for j := 1 to dist do begin
          jx := tx + (j-1) * dx; jy := ty + (j-1) * dy;
          ux := jx; uy := jy;{CCC}
          if CheckBox1.Checked then
          if (ux >= 1) and (ux <= SpaceSize) then
          if (uy >= 1) and (uy <= SpaceSize) then
          begin
          if jx <= 0       then jx := SpaceSize + jx;
          if jx >  SpaceSize then jx := jx - SpaceSize;
          if jy <= 0       then jy := SpaceSize + jy;
          if jy >  SpaceSize then jy := jy - SpaceSize;
          {rd := Int(dist);}
          rd := Sqr(Int(iAx-ux)) + Sqr(Int(iAy-uy));
          rd := Sqrt(rd);
          if rd <= MaxDist1 then begin
            if RadioGroup5.ItemIndex <= 1 then rd :=
1.0;
          if RadioGroup5.ItemIndex > 1 then
            case RadioGroup5.ItemIndex of
              2 : rd := rd;
              3 : rd := rd * rd;
              4 : rd := rd * rd * rd;
              5 : rd := rd * rd * rd * rd;
            end; { of case }
          agr_value := 1.0/rd;
          if not CheckBox5.Checked then
          begin
          Pressure[TAgent[jx,jy].opinion]
                  := Pressure[TAgent[jx,jy].opinion] +
agr_value;
          Asize[TAgent[jx,jy].opinion]
            := Asize[TAgent[jx,jy].opinion] + 1;
          end
          else
          begin
          Pressure[Agent[jx,jy].opinion]
                  := Pressure[Agent[jx,jy].opinion] +
agr_value;
          Asize[Agent[jx,jy].opinion]
            := Asize[Agent[jx,jy].opinion] + 1;
          end; { of if not CheckBox5.Checked then}
        end; { of rd <= 20.0 }
        end; { of if CheckBox1.Checked }
        if not CheckBox1.Checked then begin
        if jx <= 0       then jx := SpaceSize + jx;
        if jx >  SpaceSize then jx := jx - SpaceSize;
        if jy <= 0       then jy := SpaceSize + jy;
        if jy >  SpaceSize then jy := jy - SpaceSize;
        {rd := Int(dist);}
        rd := Sqr(Int(iAx-ux)) + Sqr(Int(iAy-uy));
        rd := Sqrt(rd);
        if rd <= MaxDist1 then begin
            if RadioGroup5.ItemIndex <= 1 then rd :=
1.0;
          if RadioGroup5.ItemIndex > 1 then
            case RadioGroup5.ItemIndex of
              2 : rd := rd;
              3 : rd := rd * rd;
              4 : rd := rd * rd * rd;
              5 : rd := rd * rd * rd * rd;
            end; { of case }
          agr_value := 1.0/rd;
          if not CheckBox5.Checked then
          begin
          Pressure[TAgent[jx,jy].opinion]
                  := Pressure[TAgent[jx,jy].opinion] +
agr_value;
          Asize[TAgent[jx,jy].opinion]
            := Asize[TAgent[jx,jy].opinion] + 1;
          end
          else
          begin
          Pressure[Agent[jx,jy].opinion]
                  := Pressure[Agent[jx,jy].opinion] +
agr_value;
                Asize[Agent[jx,jy].opinion]
                  := Asize[Agent[jx,jy].opinion] + 1;
                end; { of if not CheckBox5.Checked }
              end; { of rd <= 20.0 }
              end; { of if not CheckBox1.Checked }
            end; { of j Loop }
          end; { of i Loop }
        until dist = MaxDist;
        for i := 1 to DimSize do
          if Asize[i] > 0 then
          if not CheckBox4.Checked then
            Pressure[i] := Pressure[i]/Sqrt(Int(Asize[i]))
          else
            Pressure[i] := Pressure[i];{CCC}
      end; { of CalcPressure }

    begin
      for i := 1 to DimSize do
        Pressure[i] := 0.0;
      CalcPressure(iAx, iAy);
      maxval := Pressure[1]; maxid[1] := 1;
      for i := 2 to DimSize do
        if Pressure[i] > maxval then begin
          maxval := Pressure[i]; maxid[1] := i;
        end; { of if Pressure }
      ntie := 0;
      for i := 1 to DimSize do
        if Pressure[i] = maxval then begin
          ntie := ntie + 1;
          maxid[ntie] := i;
        end; { of if Pressure }
      if ntie = 1 then
      begin
        NewOpinion := maxid[ntie];
      end
      else
      begin
        i := random(ntie) + 1;
        NewOpinion := maxid[i];
      end; { of if ntie = 1 }
      if NewOpinion <> Agent[iAx,iAy].opinion then
        NChange := NChange + 1;
      Agent[iAx,iAy].opinion := NewOpinion;
    end; { of TryChange }


{*****  Drawing an Agent  *****}
    procedure DrawAgent(ix,iy : byte);
      var
        i, j, k   : integer;
        Gx, Gy    : integer;
        Sid, Aid  : integer;
      begin
        Gx := 8*(ix-1); Gy := 8*(iy-1);
        with MapBox.Canvas do begin
                                        Pen.Color       :=
clBlack{AgentColor(Agent[ix,iy].dim)};
          Brush.Color := AgentColor(Agent[ix,iy].opinion);
          Rectangle(Gx, Gy, Gx+8, Gy+8);
        end; { of with MapBox.Canvas }
      end; { of DrawAgent }

{*****  Drawing an Agent  *****}
    procedure DrawResults;
      var
        i, j, k   : integer;
        iAx, iAy  : byte;
        str10     : string[10];
      begin
        if not CheckBox5.Checked then
        for i := 1 to NofA do begin
          j := TurnOrder[i];
          iAx := j mod SpaceSize;
          if iAx = 0 then iAx := SpaceSize;
          iAy := (j-1) div SpaceSize + 1;
```

```
          DrawAgent(iAx, iAy);
        end; { of i Loop }
        for i := 1 to NofG do Gfreq[2,i] := Gfreq[1,i];
        for i := 1 to NofG do Gfreq[1,i] := 0;
        for i := 1 to SpaceSize do
          for j := 1 to SpaceSize do
                                    Gfreq[1,Agent[i,j].opinion]  :=
Gfreq[1,Agent[i,j].opinion] + 1;
        DrawProcess(Sender, iRound);
        for i := 1 to DimSize do Asize[i] := 0;
        for i := 1 to SpaceSize do
          for j := 1 to SpaceSize do
                                    Asize[Agent[i,j].opinion]  :=
Asize[Agent[i,j].opinion] + 1;
        for i := 1 to DimSize do begin
          str10 := IntToStr(Asize[i]);
          case i of
            1 : begin
                  Edit8.Text := str10;
                  Edit8.Refresh;
                end;
            2 : begin
                  Edit9.Text := str10;
                  Edit9.Refresh;
                end;
            3 : begin
                  Edit10.Text := str10;
                  Edit10.Refresh;
                end;
            4 : begin
                  Edit11.Text := str10;
                  Edit11.Refresh;
                end;
            5 : begin
                  Edit12.Text := str10;
                  Edit12.Refresh;
                end;
          end; { of case i }
        end; { of for i Loop }
      end; { of DrawResults }

{*****  Judgnent if round be repeated  *****}
  procedure RepeatJudgment;
    var
        i, n    : integer;
        str50   : string[50];
    begin
        if iRound >= NofRounds then end_round := true;
       { GetIndex(Entropy);}
        str50 := IntToStr(iRound) + ' : ';
        str50 := str50 + IntToStr(NChange) + '  ';
        {str50 := str50 + FloatToStrF(Entropy,ffFixed,10,4);}
        form1.memo1.lines.add(str50);
        for i := 5 downto 2 do
          N_change[i] := N_change[i-1];
        N_change[1] := NChange;
        n := 0;
        for i := 1 to 2 do n := n + N_change[i];
        if n = 0 then end_round := true;
        for i := 1 to 16 do
          if Gfreq[1,i] = SpaceSize*SpaceSize then end_round
:= true;
    end; { of RepeatJudgment }

{*****  Finale  *****}
  procedure Finale;
    begin
          form1.memo1.lines.add('Normal End of Job');
    end; { of Finale }

begin
     GlobalInit;
     SimInit;
     repeat {Round}
          RoundInit;
          for A_id := 1 to NofA do begin
            iAgent := TurnOrder[A_id];
            LoopInit;
            TryChange;
            if CheckBox5.Checked then DrawAgent(iAx, iAy);
          end; { of A_id Loop }
          DrawResults;
          RepeatJudgment;
          {if CheckBox1.Checked then Mutation;}
     until end_round;
     Finale;
end;

procedure TForm1.help1Click(Sender: TObject);
  var  str50 : string[50];
begin
     str50 := 'Social Impact simulation, ';
     form1.memo1.lines.add(str50);
     str50 := 'Multi-opinion version です．';
     form1.memo1.lines.add(str50);
     str50 := 'Start ボタンで開始，End ボタンで終了です．';
     form1.memo1.lines.add(str50);
     str50 := 'モデルを次のように選択して下さい．';
     form1.memo1.lines.add(str50);
     str50 := '・accumulative influence model:（デフォールト）';
     form1.memo1.lines.add(str50);
     str50 := '  Summation M をチェックする．';
     form1.memo1.lines.add(str50);
     str50 := '  Distance Par を４に設定．';
     form1.memo1.lines.add(str50);
     str50 := '・Fraction-size model:';
     form1.memo1.lines.add(str50);
     str50 := '  Summation M のチェックを外す．';
     form1.memo1.lines.add(str50);
     str50 := '  Distance Par を２に設定．';
     form1.memo1.lines.add(str50);
     str50 := '条件変更はラジオボタンで行います．';
     form1.memo1.lines.add(str50);
     str50 := ' Rounds: 継続ラウンド数';
     form1.memo1.lines.add(str50);
     str50 := ' Initial Dist: 初期比率';
     form1.memo1.lines.add(str50);
     str50 := '  （ピンク色のエージェントの初期比率です．
';
     form1.memo1.lines.add(str50);
     str50 := '    他のエージェントは同比率になります．';
     form1.memo1.lines.add(str50);
     str50 := ' N Opinions: 意見数';
     form1.memo1.lines.add(str50);
     str50 := ' Distance Par:「距離係数」';
     form1.memo1.lines.add(str50);
     str50 := ' Edge:チェックを外すと空間は torus';
     form1.memo1.lines.add(str50);
     str50 := ' Wide Scope:計算範囲の拡大';
     form1.memo1.lines.add(str50);
     str50 := ' Remove Self: 自己参照の解除';
     form1.memo1.lines.add(str50);
     str50 := ' Summation M:Accumulative model に変更';
     form1.memo1.lines.add(str50);
     str50 := ' ただし Distance Par を４に設定する．';
     form1.memo1.lines.add(str50);
     str50 := ' Random Seq:セルのターンを randomize';
     form1.memo1.lines.add(str50);
     str50 := '---  End of Help  ---';
     form1.memo1.lines.add(str50);

end;

end.
```

相互調整モデル

```
{ Adjustment Simulation program, Demo version          }
{              Written in Borland Delphi Ver.5.0 J   }
{  Eiji TAKAGI            2000.10.17              }
unit Adjdemoa;

interface

uses
    Windows, Messages, SysUtils, Classes, Graphics,
Controls, Forms, Dialogs,
 ExtCtrls, StdCtrls, Buttons, Menus;

type
 TForm1 = class(TForm)
  MapBox: TPaintBox;
  ProcessBox: TPaintBox;
  BitBtn1: TBitBtn;
  BitBtn2: TBitBtn;
  Panel1: TPanel;
  Edit1: TEdit;
  Edit2: TEdit;
  Memo1: TMemo;
  RadioGroup1: TRadioGroup;
  RadioGroup2: TRadioGroup;
  RadioGroup3: TRadioGroup;
  RadioGroup4: TRadioGroup;
  RadioGroup5: TRadioGroup;
  Label2: TLabel;
  Label3: TLabel;
  StaticText1: TStaticText;
  StaticText2: TStaticText;
  procedure DefaultValues(Sender: TObject);
  procedure PaintB(Sender: TObject);
  procedure PaintB1(Sender: TObject);
  procedure StrategyMap(Sender: TObject);
  procedure DrawMap(Sender: TObject);
  procedure ProcessFrame(Sender: TObject);
     procedure DrawProcess(Sender: TObject; iRound :
integer);
  procedure FormCreate(Sender: TObject);
  procedure BitBtn1Click(Sender: TObject);
  procedure BitBtn2Click(Sender: TObject);

 private
  { Private 宣言 }
 public
  { Public 宣言 }
 end;

const
     SpaceSize  = 50;
     MapSize    : array[1..2] of integer = (400, 400);
     ProcessSize : array[1..2] of integer = (220, 350);
     CellColor  : array[1..9] of TColor
                        = (clWhite, clRed, clRed{Fuchsia},
clLime{Green}, clLime{Green},
               clBlue, clBlue{Aqua}, clBlack, clBlack{Gray});
     AColor : array[1..16] of TColor
           = (clRed, clFuchsia, clYellow, clWhite, clSilver,
clLime,
                  clAqua, clBlue, clTeal, clOlive, clPurple,
clMaroon,
           clGreen, clNavy, clGray, clBlack);
     Color4 : array[1..4] of TColor
        = (clBlue, clLime{Teal}, clYellow{Blue}, clFuchsia);
     NofA       = SpaceSize*SpaceSize; { N of Agents }
 type
    boolvec   = array[1..10] of boolean;
    Cell      = record
                  dim       : boolvec;
                  payoff    : reaL;
               end;

var
     Form1      : TForm1;
     Agent      : array[1..SpaceSize, 1..SpaceSize] of Cel
     StdRed, StdGreen, StdBlack, StdBlue : boolvec;
     DimSize    : byte;
     StdMax     : array[1..2] of byte;
     NofG       : byte;
     Gfreq      : array[1..2,1..16] of integer;
      { group frequencies recorder }
      {1..2 =  1: current Round, 2: previous round }
     N_change   : array[1..5] of integer;

implementation

{$R *.DFM}

{=================================}
 function GroupId(i,j: byte) : byte;
  var
     k         : integer;
     n, bit,num  : integer;
  begin
     n := 1; num := 1;
     for k := DimSize downto 1 do begin
       if Agent[i,j].dim[k] then bit := 1 else bit := 0;
       num := num + bit*n;
       n := n*2;
     end; { of k Loop }
     GroupId := num;
  end; { of function GroupId }

 function AgentColor(x : boolvec): Tcolor;
  var
     i, j       : integer;
     n, m       : array[1..4] of byte;
     max, maxid  : byte;
     cnt        : byte;
     p, q, z    : integer;
     bit        : byte;
  begin
     if DimSize = 10 then begin
     for i := 1 to 4 do n[i] := 0;
     for i := 1 to DimSize do begin
       if x[i] = StdRed[i]   then n[1] := n[1] + 1;
       if x[i] = StdGreen[i] then n[2] := n[2] + 1;
       if x[i] = StdBlack[i] then n[3] := n[3] + 1;
       if x[i] = StdBlue[i]  then n[4] := n[4] + 1;
     end; { of i Loop }
     max := n[1]; maxid := 1;
     for i := 2 to 4 do
       if n[i] > max then begin
         max := n[i];
         maxid := i;
       end; { of if }
     cnt := 0;
     for i := 1 to 4 do
       if n[i] >= max then begin
         cnt := cnt + 1;
         m[cnt] := i;
       end; { of if }
     if cnt > 1 then begin
       j := random(cnt) + 1;
       maxid := m[j];
     end; { of if }
     if max <= StdMax[1] then
     begin
        AgentColor := CellColor[1];
     end
     else
     begin
        if max >= StdMax[2] then j := 0 else j := 1;
        AgentColor := CellColor[2+2*(maxid-1)+j];
     end; { of if max <= StdMax[1] }
     end
     else
     begin
        p := 1; q := 1;
```

```
        for i := DimSize downto 1 do begin
          if x[i] then bit := 1 else bit := 0;
          q := q + bit*p;
          p := p*2;
        end; { of i Loop }
        case DimSize of
          4 : AgentColor := AColor[q];
          2 : AgentColor := Color4[q];
          1 : AgentColor := Color4[q];
        end; { of case }
      end; { of if DimSize }
  end; { of function AgentColor }

 function AgentColor2(id : byte): Tcolor;
  var
    i, j      : integer;
    n, m      : array[1..4] of byte;
    max, maxid  : byte;
    cnt       : byte;
    p, q, z   : integer;
    bit       : byte;
  begin
        case DimSize of
          4 : AgentColor2 := AColor[id];
          2 : AgentColor2 := Color4[id];
          1 : AgentColor2 := Color4[id];
        end; { of case }
  end; { of function AgentColor2 }

{###########  Initial Values  ############}
 procedure TForm1.DefaultValues(Sender: TObject);
   var
     i, j, k, L    : integer;
     InitialDist : reaL;
   begin
     randomize;

     if RadioGroup4.ItemIndex <= 1 then begin
     case RadioGroup4.ItemIndex of
       0 : InitialDist := 0.5;
       1 : InitialDist := 0.4;
     end; { of case RadioGroup4 }
     for i := 1 to SpaceSize do
       for j := 1 to SpaceSize do begin
         for k := 1 to 10 do begin
             if random < InitialDist then Agent[i,j].dim[k] :=
true
                                 else Agent[i,j].dim[k] := false;
         end; { of k Loop }
         Agent[i,j].payoff := 0.0;
       end; { of j Loop }
     end; { of if RadioGroup4 <= 1 }

     if RadioGroup4.ItemIndex >= 2 then begin
     for i := 1 to SpaceSize do
       for j := 1 to SpaceSize do begin
         InitialDist := 0.5;
         for k := 1 to 2 do Agent[i,j].dim[k] := true;
         for k := 3 to 10 do
             if random < InitialDist then Agent[i,j].dim[k] :=
true
                                 else Agent[i,j].dim[k] := false;
         k := random(8);
         if RadioGroup4.ItemIndex = 2 then
         case k of
           3   : Agent[i,j].dim[2] := false;
           4   : Agent[i,j].dim[1] := false;
           5..7 : begin
                  Agent[i,j].dim[1] := false;
                  Agent[i,j].dim[2] := false;
                end;
         end; { of case k of }
         if RadioGroup4.ItemIndex = 3 then
         case k of
           1   : Agent[i,j].dim[2] := false;
           2..4 : Agent[i,j].dim[1] := false;
           5..7 : begin
                  Agent[i,j].dim[1] := false;
                  Agent[i,j].dim[2] := false;
                end;
         end; { of case k of }
         Agent[i,j].payoff := 0.0;
       end; { of j Loop }
     end; { of if RadioGroup4 >= 2 }

     case RadioGroup3.ItemIndex of
       0 : begin
             DimSize := 1;
             StdMax[1] := 2; StdMax[2] := 1;
           end;
       1 : begin
             DimSize := 2;
             StdMax[1] := 2; StdMax[2] := 1;
           end;
       2 : begin
             DimSize := 4;
             StdMax[1] := 3; StdMax[2] := 4;
           end;
     end; { of case RadioGroup3 }
     NofG := 1;
     for i := 1 to DimSize do NofG := NofG * 2;
     for i := 1 to DimSize do begin
       StdRed[i] := true; StdGreen[i] := false;
     end; { of i Loop }
     for i := 1 to DimSize div 2 do begin
       StdBlack[i] := true; StdBlue[i] := false;
     end; { of i Loop }
     for i := DimSize div 2 + 1 to DimSize do begin
       StdBlack[i] := false; StdBlue[i] := true;
     end; { of i Loop }
     for i := 1 to 5 do N_change[i] := 0;
   end; { of DefaultValues }

{###########  Landscape Map  ############}
 procedure TForm1.StrategyMap(Sender: TObject);
   var
     i, j, k  : integer;
     InSt     : string[15];
   begin
     with MapBox.Canvas do begin
       Pen.Color := clBlack; Brush.Color:= clBlack;
       Rectangle(0, 0, Width, Height);
     end; { of with MapBox.Canvas }
   end; { of StrategyMap }

{##########  Drawing Landscape Map  ###########}
 procedure TForm1.DrawMap(Sender: TObject);
   var
     i, j  : byte;

   procedure DrawPlayer(ix,iy : byte);
     var
       i, j, k   : integer;
       Gx, Gy    : integer;
       Sid, Aid  : integer;
     begin
       Gx := 8*(ix-1); Gy := 8*(iy-1);
       with MapBox.Canvas do begin
                                       Pen.Color
clBlack{AgentColor(Agent[ix,iy].dim)};
         Brush.Color := AgentColor(Agent[ix,iy].dim);
         Rectangle(Gx, Gy, Gx+8, Gy+8);
       end; { of with MapBox.Canvas }
     end; { of DrawPlayer }

   begin
     for i := 1 to SpaceSize do
       for j := 1 to SpaceSize do DrawPlayer(i,j);
   end; { of DrawMap }

  procedure TForm1.ProcessFrame(Sender: TObject);
   begin
     with ProcessBox.Canvas do begin
       Pen.Color := clBlack; Brush.Color:= clWhite;
```

```
        Rectangle(0, 0, Width, Height);
          Font.Name := 'Times New Roman'; Font.Style :=
[fsBold];
        Font.Color := clNavy; Font.Size := 11;
            TextOut(ProcessSize[1] div 2 -50, 2, 'Evolution
Process');
        Pen.Color := clBlack;
        MoveTo(10, 30); LineTo(10, ProcessSize[2]-20);
                          MoveTo(10,  ProcessSize[2]-20);
LineTo(ProcessSize[1]-10, ProcessSize[2]-20);
        Font.Color := clBlack; Font.Size := 10;
        TextOut(10,16, 'frequency');
                        TextOut(ProcessSize[1] div 2-30,
ProcessSize[2]-15, 'Round');
      end; { of with ProcessBox.Canvas }
   end; { of ProcessFrame }

{***** Drawing the Evolution Process per Iteration *****}
  procedure TForm1.DrawProcess(Sender: TObject; iRound
: integer);
   const
      Ux = 2;
      Uy = 10;
   var
      i, j, k      : byte;
      ipage, jround : integer;
      CumCnt       : array[1..2] of integer;
      Gxy          : array[1..2,1..2] of integer;
      xy           : array[1..4] of TPoint;
      NofP         : integer;
   begin
      NofP := SpaceSize*SpaceSize;
      ipage  := (iRound-1) div 100 + 1;
      jround := iRound mod 100; if jround = 0 then jround :=
100;
      if jround = 1 then begin
        with ProcessBox.Canvas do begin
          Pen.Color := clWhite; Brush.Color:= clWhite;
                      Rectangle(10, 30, ProcessSize[1]-10,
ProcessSize[2]-20);
          Pen.Color := clBlack;
          MoveTo(10, 30); LineTo(10, ProcessSize[2]-20);
          MoveTo(10, ProcessSize[2]-20);
          LineTo(ProcessSize[1]-10, ProcessSize[2]-20);
        end; { of with ProcessBox.Canvas }
      end; { of if }
      for i := 1 to 2 do CumCnt[i] := 0;
      for i := NofG downto 1 do
        if (Gfreq[1,i]>0) or (Gfreq[2,i]>0) then begin
          Gxy[1,1] := 10 + (jround-1)*Ux;
          Gxy[2,1] := 10 + jround*Ux;
              Gxy[1,2] := ProcessSize[2]-20-NofP div Uy +
CumCnt[2] div Uy;
              Gxy[2,2] := ProcessSize[2]-20-NofP div Uy +
CumCnt[1] div Uy;
          xy[1] := Point(Gxy[1,1], Gxy[1,2]);
          xy[2] := Point(Gxy[2,1], Gxy[2,2]);
              for j := 1 to 2 do CumCnt[j] := CumCnt[j] +
Gfreq[j,i];
              Gxy[1,2] := ProcessSize[2]-20-NofP div Uy +
CumCnt[2] div Uy;
              Gxy[2,2] := ProcessSize[2]-20-NofP div Uy +
CumCnt[1] div Uy;
          xy[4] := Point(Gxy[1,1], Gxy[1,2]);
          xy[3] := Point(Gxy[2,1], Gxy[2,2]);
          with ProcessBox.Canvas do begin
              Pen.Color := AgentColor2(i); Brush.Color :=
AgentColor2(i);
            Polygon(xy);
          end; { of with ProcessBox.Canvas }
        end; { of if }
   end; { of DrawProcess }

{########### Form Paint ############}
procedure TForm1.PaintB(Sender: TObject);
 begin
      MapBox.Width  := MapSize[1];  MapBox.Height  :=
MapSize[2];
    with MapBox.Canvas do begin
      Brush.Color := clWindow;
      Rectangle(0, 0, Width, Height);
      Pen.Color := clWindow;
    end; { of with MapBox }
 end; { of PaintB }

{########### Form Paint 1 ############}
procedure TForm1.PaintB1(Sender: TObject);
 begin
      MapBox.Width  := MapSize[1];  MapBox.Height  :=
MapSize[2];
    with MapBox.Canvas do begin
      Brush.Color := clBlack;
      Rectangle(0, 0, Width, Height);
      Pen.Color := clWindow;
    end; { of with MapBox }
    DefaultValues(Sender);
    StrategyMap(Sender);
    DrawMap(Sender);
    ProcessFrame(Sender);
 end; { of PaintB1 }

{########### Form Create ############}
procedure TForm1.FormCreate(Sender: TObject);
begin
   PaintB1(Sender);
end;

{########### Close Button ############}
procedure TForm1.BitBtn2Click(Sender: TObject);
begin
   close;
end;

{########### Main Routine ############}
procedure TForm1.BitBtn1Click(Sender: TObject);

 var
    iSim      : integer;  { counter of Sim iteration }
    iRound    : integer;  { counter of round }
    NofRounds : integer;
    iAgent    : integer;
    iAx, iAy  : byte;
    A_id      : integer;
    P_try     : reaL;
    NChange   : integer;
    end_round : boolean;
    TurnOrder : array[1..NofA] of integer;
    Entropy   : reaL;

 procedure GetIndex(var entrp : reaL);
  var
    Ientropy  : array[0..1023] of integer;
    i, j, k   : integer;
    n, bit,num : integer;
    x, y      : reaL;
  begin
    for i := 0 to 1023 do Ientropy[i] := 0;
    for i := 1 to SpaceSize do
      for j := 1 to SpaceSize do begin
        n := 1; num := 0;
        for k := DimSize downto 1 do begin
          if Agent[i,j].dim[k] then bit := 1 else bit := 0;
          num := num + bit*n;
          n := n*2;
        end; { of k Loop }
        Ientropy[num] := Ientropy[num] + 1;
      end; { of j Loop }
    y := 0.0;
    for i := 0 to 1023 do
      if Ientropy[i] > 0 then begin
        x := Int(Ientropy[i])/Int(NofA);
        y := y - x * Ln(x);
      end; { of i Loop }
    entrp := y;
```

```
    end; { of GetIndex }

{*****  Global Initialization  *****}
  procedure GLobaLInit;
    begin
       Randomize;
       case RadioGroup1.ItemIndex of
        0 : NofRounds :=  50;
        1 : NofRounds := 100;
        2 : NofRounds := 150;
        3 : NofRounds := 500;
       end; { of case RadioGroup1 }
       case RadioGroup2.ItemIndex of
        0 : P_try := 0.1;
        1 : P_try := 0.2;
        2 : P_try := 0.5;
       end; { of case RadioGroup1 }
    end; { of GLobaLInit }

{*****  Simulation Run Initialization  *****}
  procedure SimInit;
    var
       i, j, k, L  : integer;
       str50   : string[50];
    begin
       DefaultValues(Sender);
       StrategyMap(Sender);
       DrawMap(Sender);
       ProcessFrame(Sender);
       end_round := false;
       iRound := 0;
       GetIndex(Entropy);
       str50 := IntToStr(iRound) + ' : ';
       str50 := str50 + FloatToStrF(Entropy,ffFixed,10,4);
       form1.memo1.lines.add(str50);
       for i := 1 to NofG do Gfreq[1,i] := 0;
       for i := 1 to SpaceSize do
         for j := 1 to SpaceSize do
           Gfreq[1,GroupId(i,j)] := Gfreq[1,GroupId(i,j)] + 1;
    end; { of SimInit }

{*****  Round Initialization  *****}
  procedure RoundInit;

    procedure PutTurnOrder;
      var
        i, j, k    : integer;
        seL1, seL2 : integer;
      begin
         for i := 1 to NofA do TurnOrder[i] := i;
         for i := 1 to 10 do
           for j := 1 to NofA do begin
                          seL1  :=  random(NofA)+1;  seL2  :=
random(NofA)+1;
             k := TurnOrder[seL1];
             TurnOrder[seL1] := TurnOrder[seL2];
             TurnOrder[seL2] := k;
           end; { of j Loop }
      end; { of PutTurnOrder }

    begin
       iRound := iRound + 1;
       Edit1.Text := IntToStr(iRound);
       Edit1.Refresh;
       PutTurnOrder;
       NChange := 0;
    end; { of RoundInit }

{*****  Loop Initialization  *****}
  procedure LoopInit;
    begin
       Edit2.Text := IntToStr(A_id);
       Edit2.Refresh;
       iAx := iAgent mod SpaceSize;
       if iAx = 0 then iAx := SpaceSize;
       iAy := (iAgent-1) div SpaceSize + 1;
    end; { of LoopInit }

{*****  Agent's Trial to Change  *****}
  procedure TryChange;
    var
       i, j   : integer;
       MemoryPayoff : reaL{byte};
       MemoryDim    : boolvec;

    function CountAgreement(ix, iy : byte): byte;
      var
         i, j, k, n   : byte;
         x1, y1       : byte;
      begin
         n := 0;
         x1 := ix;
         y1 := iy - 1;
         if y1 < 1 then y1 := SpaceSize;
         for i := 1 to DimSize do
           if Agent[ix,iy].dim[i] = Agent[x1,y1].dim[i] then n := n
+ 1;
         y1 := iy + 1;
         if y1 > SpaceSize then y1 := 1;
         for i := 1 to DimSize do
           if Agent[ix,iy].dim[i] = Agent[x1,y1].dim[i] then n := n
+ 1;
         y1 := iy;
         x1 := ix - 1;
         if x1 < 1 then x1 := SpaceSize;
         for i := 1 to DimSize do
           if Agent[ix,iy].dim[i] = Agent[x1,y1].dim[i] then n := n
+ 1;
         x1 := ix + 1;
         if x1 > SpaceSize then x1 := 1;
         for i := 1 to DimSize do
           if Agent[ix,iy].dim[i] = Agent[x1,y1].dim[i] then n := n
+ 1;
         CountAgreement := n;
      end; { of function CountAgreement }

     function CountAgreement2(ix, iy : byte): reaL;
      var
         i, j, k   : integer;
         n, dist   : byte;
         tx, ty    : integer;
         jx, jy    : integer;
         dx, dy    : integer;
         rd, rd1   : reaL;
         agr_value : reaL;
         MaxDist   : byte;
      begin
         MaxDist := 24;
         n := 0;
         rd := 0.84;
         for i := 1 to DimSize do
           if MemoryDim[i] = Agent[iAx,iAy].dim[i] then n := n
+ 1;
         if RadioGroup5.ItemIndex <= 1 then rd := 1.0;
         if RadioGroup5.ItemIndex > 1 then
           case RadioGroup5.ItemIndex of
            2 : rd := rd;
            3 : rd := rd * rd;
            4 : rd := rd * rd * rd;
            5 : rd := rd * rd * rd * rd;
           end; { of case }
         agr_value := Int(n)/Int(DimSize)/rd;
         if RadioGroup5.ItemIndex = 0 then begin
           agr_value := 0.0;
           MaxDist := 1;
         end; { of if }
         dist := 0;
         repeat
            dist := dist + 1;
            for i := 1 to 4 do begin
              case i of
                1 : begin
                      tx := ix - dist; ty := iy;
                      dx := 1; dy := 1;
```

```
                end;
              2 : begin
                    tx := ix; ty := iy + dist;
                    dx := 1; dy := -1;
                end;
              3 : begin
                    tx := ix + dist; ty := iy;
                    dx := -1; dy := -1;
                end;
              4 : begin
                    tx := ix; ty := iy - dist;
                    dx := -1; dy := 1;
                end;
            end; { of case i }
            for j := 1 to dist do begin
              jx := tx + (j-1) * dx; jy := ty + (j-1) * dy;
              if jx <= 0        then jx := SpaceSize + jx;
              if jx >  SpaceSize then jx := jx - SpaceSize;
              if jy <= 0        then jy := SpaceSize + jy;
              if jy >  SpaceSize then jy := jy - SpaceSize;
              n := 0;
              for k := 1 to DimSize do
                if Agent[ix,iy].dim[k] = Agent[jx,jy].dim[k]
                then n := n + 1;
              rd := Int(dist);
                 if RadioGroup5.ItemIndex <= 1 then rd :=
1.0;
              if RadioGroup5.ItemIndex > 1 then
                case RadioGroup5.ItemIndex of
                  2 : rd := rd;
                  3 : rd := rd * rd;
                  4 : rd := rd * rd * rd;
                  5 : rd := rd * rd * rd * rd;
                end; { of case }
              agr_value := agr_value
                        + Int(n)/Int(DimSize)/rd;
            end; { of j Loop }
          end; { of i Loop }
        until dist = MaxDist;
        CountAgreement2 := agr_value;
      end; { of function CountAgreement2 }

    begin
      for i := 1 to DimSize do
        MemoryDim[i] := Agent[iAx,iAy].dim[i];
      MemoryPayoff := Agent[iAx,iAy].payoff;
      { if iRound = 1 then begin}
        MemoryPayoff := CountAgreement2(iAx,iAy);
      {end;} { of if iRound = 1 }
      for i := 1 to DimSize do
        if random < P_try then
          Agent[iAx,iAy].dim[i] := not Agent[iAx,iAy].dim[i];
      Agent[iAx,iAy].payoff := CountAgreement2(iAx, iAy);
      if Agent[iAx,iAy].payoff > MemoryPayoff then
      begin
        NChange := NChange + 1;
      end
      else
      begin
        for i := 1 to DimSize do
          Agent[iAx,iAy].dim[i] := MemoryDim[i];
        Agent[iAx,iAy].payoff := MemoryPayoff;
      end; { of if }
    end; { of TryChange }


{***** Drawing an Agent *****}
  procedure DrawAgent(ix,iy : byte);
    var
      i, j, k   : integer;
      Gx, Gy    : integer;
      Sid, Aid  : integer;
    begin
      Gx := 8*(ix-1); Gy := 8*(iy-1);
      with MapBox.Canvas do begin
                                          Pen.Color          :=
clBlack{AgentColor(Agent[ix,iy].dim)};
        Brush.Color := AgentColor(Agent[ix,iy].dim);
        Rectangle(Gx, Gy, Gx+8, Gy+8);
      end; { of with MapBox.Canvas }
    end; { of DrawAgent }

{***** Drawing an Agent *****}
  procedure DrawResults;
    var
      i, j, k   : integer;
    begin
      for i := 1 to NofG do Gfreq[2,i] := Gfreq[1,i];
      for i := 1 to NofG do Gfreq[1,i] := 0;
      for i := 1 to SpaceSize do
        for j := 1 to SpaceSize do
          Gfreq[1,GroupId(i,j)] := Gfreq[1,GroupId(i,j)] + 1;
      DrawProcess(Sender, iRound);
    end; { of DrawResults }

{***** Judgnent if round be repeated *****}
  procedure RepeatJudgment;
    var
      i, n    : integer;
      str50   : string[50];
    begin
      if iRound >= NofRounds then end_round := true;
      GetIndex(Entropy);
      str50 := IntToStr(iRound) + ' : ';
      str50 := str50 + IntToStr(NChange) + '  ';
      str50 := str50 + FloatToStrF(Entropy,ffFixed,10,4);
      form1.memo1.lines.add(str50);
      for i := 5 downto 2 do
        N_change[i] := N_change[i-1];
      N_change[1] := NChange;
      n := 0;
      for i := 1 to 3 do n := n + N_change[i];
      if n = 0 then end_round := true;
      for i := 1 to 16 do
        if Gfreq[1,i] = SpaceSize*SpaceSize then end_round
:= true;
    end; { of RepeatJudgment }

{***** Finale *****}
  procedure Finale;
    begin
        form1.memo1.lines.add('Normal End of Job');
    end; { of Finale }

begin
    GlobalInit;
    SimInit;
    repeat {Round}
      RoundInit;
      for A_id := 1 to NofA do begin
        iAgent := TurnOrder[A_id];
        LoopInit;
        TryChange;
        DrawAgent(iAx, iAy);
      end; { of A_id Loop }
      DrawResults;
      RepeatJudgment;
    until end_round;
    Finale;
end;

end.
```

限定交換モデル

```
{ Restricted Exchange Simulation Program, Revisited        }
{              Written in Borland Delphi Ver.6 J      }
{ Reciprocity Norm Simulation                          }
{     Recording version for a homogeneous population
'R_ExR1' }
{  Eiji TAKAGI            2001.09.18                   }
unit R_exR1a;

interface

uses
    Windows, Messages, SysUtils, Classes, Graphics,
Controls, Forms, Dialogs,
  ExtCtrls, StdCtrls, Buttons, Menus;

type
  TForm1 = class(TForm)
    InfoBox: TPaintBox;
    MapBox: TPaintBox;
    ProcessBox: TPaintBox;
    BitBtn1: TBitBtn;
    BitBtn2: TBitBtn;
    Panel1: TPanel;
    Edit1: TEdit;
    Edit2: TEdit;
    Memo1: TMemo;
    RadioGroup1: TRadioGroup;
    RadioGroup2: TRadioGroup;
    RadioGroup3: TRadioGroup;
    Label2: TLabel;
    Label3: TLabel;
    StaticText1: TStaticText;
    StaticText2: TStaticText;
    Image1: TImage;
    CheckBox1: TCheckBox;
    EventBox: TPaintBox;
    Edit3: TEdit;
    Label1: TLabel;
    Edit4: TEdit;
    Edit5: TEdit;
    Edit6: TEdit;
    Edit7: TEdit;
    Edit8: TEdit;
    Edit9: TEdit;
    Edit10: TEdit;
    Edit11: TEdit;
    Edit12: TEdit;
    Edit13: TEdit;
    Edit14: TEdit;
    Edit15: TEdit;
    Edit16: TEdit;
    Edit17: TEdit;
    Edit18: TEdit;
    Edit19: TEdit;
    RadioGroup4: TRadioGroup;
    StaticText3: TStaticText;
    Edit20: TEdit;
    Label4: TLabel;
    Edit21: TEdit;
    Label5: TLabel;
    Edit22: TEdit;
    Label6: TLabel;
    procedure DefaultValues(Sender: TObject);
    procedure PaintB(Sender: TObject);
    procedure PaintB1(Sender: TObject);
    procedure InfoFrame(Sender: TObject);
    procedure StrategyMap(Sender: TObject);
    procedure DrawMap(Sender: TObject);
    procedure ProcessFrame(Sender: TObject);
    procedure EventFrame(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure BitBtn1Click(Sender: TObject);
    procedure BitBtn2Click(Sender: TObject);
{   procedure RadioGroup1Click(Sender: TObject);
    procedure RadioGroup2Click(Sender: TObject);
    procedure Button1Click(Sender: TObject);}
  private
    { Private 宣言 }
  public
    { Public 宣言 }
  end;

const
      MapSize     : array[1..2] of integer = (250, 250);
      ProcessSize : array[1..2] of integer = (400, 250);
      InfoBoxSize : array[1..2] of integer = (400, 200);
      EventSize   : array[1..2] of integer = (5, 5);
      CellColor   : array[0..5] of TColor =
                          { (clWhite, clYellow, clLime, clGreen,
clPurple, clOlive);}
                    (clBlack, clGray, clSilver, clWhite, clPurple,
clOlive);
      AgentColor  : array[1..6] of TColor =
                     (clRed, clBlue, clLime, clPurple, clYellow,
clTeal);
      EventColor  : array[1..4] of TColor =
                   (clRed, clBlue, clLime, clFuchsia);
      CellStyle   : array[0..5] of TBrushStyle =
                    (bsClear, bsVertical, bsCross, bsDiagCross,
bsSolid, bsBDiagonal);
      Gf          : array[1..3, 1..2, 1..2] of integer
                     = ((( 50,100), (450, 500)), ((550,100), (750,
300)),
                    ((550,400), (950,600)));
      NofP        = 100; { N of players }
      NofStr      =  22; { N of Strategies }
     { NofStr2    =  27;} { N of Strategies per Group}
      NofSG       =   6; { N of Strategy Groups }
        LossW         = 0.75; { Loss Weight of Between-Clan
transfer }
      Pmutant     = 0.015;  { Pr of Mutant Emergence }
      NofIte      = 200;  { Number of iterations for each run }
      discount    = 0.95; { discount factor weight }
      v_f         = 2.0;  { unit value of other's favor }
      Ugive       = 1.0;  { unit amount of giving }
    {  GroupLabel  : array[1..2] of string[1] = ('A', 'B');}
        Scolor        : array[1..NofStr] of TColor = (clFuchsia,
clYellow, clGreen,
                     clAqua, clWhite, clWhite, clWhite, clWhite,
clWhite,
                   clWhite, clMaroon, clGreen, clGreen, clGreen,
clGreen,
                     clGreen, clOlive, clGreen, clGreen, clGreen,
clGreen,
                   clSilver);
          Ccolor        : array[1..NofStr] of TColor = (clBlack,
clBlack, clWhite,
                      clBlack, clBlack, clBlack, clBlack, clBlack,
clBlack,
                      clBlack, clWhite, clWhite, clWhite, clWhite,
clWhite,
                      clWhite, clBlack, clWhite, clWhite, clWhite,
clWhite,
                   clBlack);
      Smark       : array[1..NofStr] of string[1]
                 = ('S','D','R','T',
                    '1','2','3','4','5','6','1','2','3','4','5','6','7',
                    '8','9','0','A','B');
      Slabel      : array[1..NofStr] of string[7]
                 = ('Saint', 'NonCoop', 'Recp', 'TFT', 'S1', 'S2',
                    'S3','S4','S5','S6','T1','T2','T3','T4','T5','T6',
                    'T7','T8','T9','T10','T11','T12');
  type
     a_vec     = array[1..8]  of byte;
     c_vec     = array[1..10] of byte;
     dig3      = array[1..3]  of byte;
     dig2      = array[1..2]  of byte;
     Agent_type = record
                    min        : dig2;
                    max        : dig2;
```

```
                ex_strategy : byte;
              end;

var
    Form1      : TForm1;
  {=== variables first determined ===}
    INofP      : array[1..NofStr] of integer;
    NofRuns    : integer; { Number of Runs per simulation
}
    Agent      : array[1..NofP] of Agent_type;
  {=== Important variables ===}
    {Strategy  : array[1..NofP] of integer;}
    Sfreq      : array[1..2, 1..NofStr] of integer;
     { strategy frequencies recorder }
  {=== Condition Variables ===}
    DoMutant   : boolean; { Introduce Mutants }
    Std_Payoff : array[1..3,1..2] of byte; { Standard payoffs
}
    iIniD      : byte;
implementation

{$R *.DFM}

{*** Converting digit vector into byte number ***}
 function DtoB(x : dig3) : byte;
  var
    i, j, k, L   : integer;
    inc          : integer;
  begin
    j := 0; inc := 1;
    for i := 1 to 3 do begin
      if x[4-i] = 1 then j := j + inc;
      inc := inc *2;
    end; { of i loop }
    DtoB := j;
  end; { of DtoN }

{***** Random Number Generator: Normal Distribution
*****}
 function Normal(m, s: real) : real;
{ inputs : m(mean) & s(standard deviation)        }
  var
    u, v   : array[1..2] of real;
    w, y   : real;
    i      : integer;
  begin
    w := 0.0;
    repeat
      for i := 1 to 2 do begin
        u[i] := random;
        v[i] := 2.0 * u[i] - 1.0;
      end; { of i loop }
      w := v[1]*v[1] + v[2]*v[2];
    until w <= 1.0;
    y := Sqrt(-2.0*Ln(w)/w);
    for i := 1 to 2 do
      u[i] := v[i]*y;
    Normal := u[1]*s + m;
  end; { of Normal }

procedure TForm1.DefaultValues(Sender: TObject);
  var
    i, j, k    : integer;
    cnt, scnt  : integer;
    iNofRuns   : integer;
    iSimCond   : integer;
    iUncertain : integer;

  procedure Shuffle;
   var
     mintemp : dig2;
     maxtemp : dig2;
     ex_stemp: byte;
     i, j    : integer;
   begin
     for i := NofP downto 2 do begin
       j := trunc(random*i) + 1;
       ex_stemp := Agent[i].ex_strategy;
       mintemp  := Agent[i].min;
       maxtemp  := Agent[i].max;
       Agent[i].ex_strategy := Agent[j].ex_strategy;
       Agent[i].min := Agent[j].min;
       Agent[i].max := Agent[j].max;
       Agent[j].ex_strategy := ex_stemp;
       Agent[j].min := mintemp;
       Agent[j].max := maxtemp;
     end; { of i Loop }
   end; { of Shuffle }

begin
  iSimCond := RadioGroup1.ItemIndex + 1;
  for i := 1 to NofStr do INofP[i] := 0;
  case iIniD{iSimCond} of
   1 : begin
         INofP[2] := 95; INofP[4] :=  5;
       end;
   2 : begin
         INofP[2] := 90; INofP[4] := 10;
       end;
   {3 : begin
         INofP[2] := 80; INofP[4] := 20;
       end;}
   3 : begin
         INofP[1] :=  0; INofP[2] := 50;
         INofP[4] := 50;
       end;
  end; { of case iSimCond }
  cnt := 0;
  for i := 1 to NofStr do begin
    if INofP[i] > 0 then
    for j := 1 to INofP[i] do begin
      cnt := cnt + 1;
      Agent[cnt].ex_strategy := i;
      for k := 1 to 2 do begin
        if random < 0.5 then Agent[cnt].min[k] := 1
                        else Agent[cnt].min[k] := 0;
        if random < 0.5 then Agent[cnt].max[k] := 1
                        else Agent[cnt].max[k] := 0;
      end; { of k Loop }
    end; { of j loop }
  end; { of i loop }
  Shuffle;
  {NNNNNN}
  {for i := 1 to NofP div 2 do
    for j := 1 to 2 do begin
      Agent[i].min[j] := 1;
      Agent[i].min[j] := 1;
      Agent[i].max[j] := 1;
      Agent[i].max[j] := 1;
    end;}
  {NNNNNN}
  for i := 1 to NofStr do Sfreq[1,i] := 0;
  for i := 1 to NofP do
                          Sfreq[1,Agent[i].ex_strategy]   :=
Sfreq[1,Agent[i].ex_strategy] + 1;
    { GetGfreq(1);}
    { iNofRuns := RadioGroup2.ItemIndex + 1;
     case iNofRuns of
      1 : NofRuns :=  50;
      2 : NofRuns := 100;
      3 : NofRuns := 200;
      4 : NofRuns := 500;
     end;} { of case iNofRuns }
    { case RadioGroup3.ItemIndex of
      0 : Std_Payoff[1] := 20;
      1 : Std_Payoff[1] := 10;
      2 : Std_Payoff[1] :=  4;
     end;} { of case iUncertain }
     {case RadioGroup4.ItemIndex of
      0 : Std_Payoff[2] := 20;
      1 : Std_Payoff[2] := 10;
      2 : Std_Payoff[2] :=  4;
     end;} { of case iUncertain } {Treatment}
     Std_Payoff[1,1] := 20; Std_Payoff[1,2] := 20;
```

```
        Std_Payoff[2,1] := 10; Std_Payoff[2,2] := 10;
        Std_Payoff[3,1] :=  4; Std_Payoff[3,2] :=  4;
    end; { of DefaultValues }

procedure TForm1.StrategyMap(Sender: TObject);
  var
     i, j, k  : integer;
     InSt     : string[15];
  begin
     with MapBox.Canvas do begin
       Pen.Color := clBlack; Brush.Color:= clWhite;
       Rectangle(0, 0, Width, Height);
        Font.Color := clNavy; Font.Size := 12; Font.Style :=
[fsBold];
       TextOut(70{Width div 2 - 100}, 3, 'Strategy Map');
     end; { of with MapBox.Canvas }
  end; { of StrategyMap }

{*****  Drawing Strategy Map & Displaying Strategy
Frequencies *****}
 procedure TForm1.DrawMap(Sender: TObject);
  var
     i, j, k   : integer;
     InSt      : string[10];
     n1        : integer;

  procedure DrawPlayer(pL : integer);
   var
     i, j, k   : integer;
     Gx, Gy    : integer;
     Sid       : integer;
   begin
      i := pL mod 10;
      if i = 0 then i := 10;
      Gx := 25 + 20*(i-1);
      j := (pL-1) div 10 + 1;
      Gy := 25 + 20*(j-1);
      Sid := Agent[pL].ex_strategy;
      with MapBox.Canvas do begin
           Pen.Color := clNavy; Font.Color := Ccolor[Sid];
Font.Size := 8;
        Brush.Color := Scolor[Sid]; Font.Style := [fsBold];
        Rectangle(Gx, Gy, Gx+20, Gy+20);
        TextOut(Gx+5, Gy+2, Smark[Sid]);
      end; { of with MapBox.Canvas }
   end; { of DrawPlayer }

  begin
     with InfoBox.Canvas do begin
     Pen.Color := clWhite;
       Brush.Color := clWhite; Font.Size := 10;
  {    for i := 1 to 2 do begin}
        n1 := 0;
        for j := 1 to 5 do begin
          i := 5; if j = 5 then i := 2;
          for k := 1 to i do begin
            n1 := n1 + 1;
            InSt := IntToStr(Sfreq[1,n1]);
            if INofP[n1] <> 0 then Font.Color := clBlack
                      else Font.Color := clGray;
            Rectangle(10+(k-1)*80, 60+70*(j-1),
              10+k*80, 60+70*(j-1)+15);
            TextOut(10+(k-1)*80, 60+70*(j-1), InSt);
          end; { of k loop }
        end; { of j loop }
     {  end;} { of i loop }
       for i := 1 to NofP do DrawPlayer(i);
     end; { of with InfoBox.Canvas }
   end; { of DrawMap }

procedure TForm1.InfoFrame(Sender: TObject);
  var
     i, j, k  : integer;
     InSt     : string[15];
     n1       : integer;
  begin
     with InfoBox.Canvas do begin
       Pen.Color := clBlack; Brush.Color:= clWhite;
       Rectangle(0, 0, Width, Height);
       {Font.Name := 'Arial';} Font.Style := [fsBold];
       Font.Color := clNavy; Font.Size := 11;
          TextOut(InfoBoxSize[1] div 2 -70, 2, '< Strategy
Distribution >');
       Font.Size := 10;
     {  for i := 1 to 2 do begin}
        n1 := 0;
        for j := 1 to 5 do begin
          i := 5; if j = 5 then i := 2;
          for k := 1 to i do begin
            n1 := n1 + 1;
                 Brush.Color := Scolor[n1]; Font.Color :=
Ccolor[n1];
            TextOut(5+(k-1)*80, 40+70*(j-1), SLabel[n1]);
            InSt := IntToStr(INofP[n1]);
            InSt := '('+InSt+')';
            Brush.Color := clWhite;
            if INofP[n1] <> 0 then Font.Color := clBlack
                      else Font.Color := clGray;
            TextOut(5+(k-1)*80, 75+70*(j-1), InSt);
          end; { of k loop }
        end; { of j loop }
     {  end; }{ of i loop }
     end; { of with InfoBox.Canvas }
  end; { of InfoFrame }

  procedure TForm1.ProcessFrame(Sender: TObject);
   begin
      with ProcessBox.Canvas do begin
        Pen.Color := clBlack; Brush.Color:= clWhite;
        Rectangle(0, 0, Width, Height);
          {Font.Name := 'Times New Roman';} Font.Style :=
[fsBold];
        Font.Color := clNavy; Font.Size := 11;
           TextOut(ProcessSize[1] div 2 -50, 2, 'Evolution
Process');
        Pen.Color := clBlack;
        MoveTo(10, 30); LineTo(10, ProcessSize[2]-20);
                      MoveTo(10,  ProcessSize[2]-20);
LineTo(ProcessSize[1]-10, ProcessSize[2]-20);
        Font.Color := clBlack; Font.Size := 10;
        TextOut(10,16, 'frequency');
                     TextOut(ProcessSize[1]  div  2-30,
ProcessSize[2]-15, 'Generation');
      end; { of with ProcessBox.Canvas }
   end; { of ProcessFrame }

  procedure TForm1.EventFrame(Sender: TObject);
   begin
      with EventBox.Canvas do begin
        Pen.Color := clBlack; Brush.Color:= clWhite;
        Rectangle(0, 0, Width, Height);
          {Font.Name := 'Times New Roman';} Font.Style :=
[fsBold];
        Font.Color := clNavy; Font.Size := 11;
        TextOut(EventSize[1] div 2 -50, 2, 'Event Types');
        Pen.Color := clBlack;
        MoveTo(10, 30); LineTo(10, EventSize[2]-20);
                      MoveTo(10,  EventSize[2]-20);
LineTo(EventSize[1]-10, EventSize[2]-20);
        Font.Color := clBlack; Font.Size := 10;
        TextOut(10,16, 'frequency');
           TextOut(EventSize[1] div 2-30, EventSize[2]-15,
'Generation');
      end; { of with EventBox.Canvas }
   end; { of EventFrame }

procedure TForm1.PaintB(Sender: TObject);
  begin
       InfoBox.Width := InfoBoxSize[1]; InfoBox.Height :=
InfoBoxSize[2];
     with InfoBox.Canvas do begin
       Brush.Color := clWindow;
       Rectangle(0, 0, Width, Height);
       Pen.Color := clWindow;
```

```
    end; { of with InfoBox }
      MapBox.Width  := MapSize[1];  MapBox.Height  :=
MapSize[2];
    with MapBox.Canvas do begin
      Brush.Color := clWindow;
      Rectangle(0, 0, Width, Height);
      Pen.Color := clWindow;
    end; { of with MapBox }
    ProcessBox.Width  := ProcessSize[1];
    ProcessBox.Height := ProcessSize[2];
    with ProcessBox.Canvas do begin
      Brush.Color := clWindow;
      Rectangle(0, 0, Width, Height);
      Pen.Color := clWindow;
    end; { of with ProcessBox }
    EventBox.Width  := EventSize[1];
    EventBox.Height := EventSize[2];
    with EventBox.Canvas do begin
      Brush.Color := clWindow;
      Rectangle(0, 0, Width, Height);
      Pen.Color := clWindow;
    end; { of with EventBox }
 end; { of PaintB }

procedure TForm1.PaintB1(Sender: TObject);
 begin
      InfoBox.Width := InfoBoxSize[1]; InfoBox.Height :=
InfoBoxSize[2];
    with InfoBox.Canvas do begin
      Brush.Color := clWindow;
      Rectangle(0, 0, Width, Height);
      Pen.Color := clWindow;
        {Font.Name := 'Times New Roman'; Font.Style :=
[fsBold];}
    end; { of with InfoBox }
      MapBox.Width  := MapSize[1];  MapBox.Height  :=
MapSize[2];
    with MapBox.Canvas do begin
      Brush.Color := clWindow;
      Rectangle(0, 0, Width, Height);
      Pen.Color := clWindow;
    end; { of with MapBox }
    ProcessBox.Width  := ProcessSize[1];
    ProcessBox.Height := ProcessSize[2];
    with ProcessBox.Canvas do begin
      Brush.Color := clWindow;
      Rectangle(0, 0, Width, Height);
      Pen.Color := clWindow;
    end; { of with ProcessBox }
    EventBox.Width  := EventSize[1];
    EventBox.Height := EventSize[2];
    with EventBox.Canvas do begin
      Brush.Color := clWindow;
      Rectangle(0, 0, Width, Height);
      Pen.Color := clWindow;
    end; { of with EventBox }
    DefaultValues(Sender);
    StrategyMap(Sender);
    ProcessFrame(Sender);
    InfoFrame(Sender);
    DrawMap(Sender);
    EventFrame(Sender);
 end; { of PaintB1 }

procedure TForm1.FormCreate(Sender: TObject);
begin
   PaintB1(Sender);
end;

{
procedure TForm1.RadioGroup1Click(Sender: TObject);
begin
   PaintB(Sender);
   InfoFrame(Sender);
   StrategyMap(Sender);
   ProcessFrame(Sender);
end;

procedure TForm1.RadioGroup2Click(Sender: TObject);
begin
   PaintB(Sender);
   InfoFrame(Sender);
   StrategyMap(Sender);
   ProcessFrame(Sender);
end;
}

procedure TForm1.BitBtn2Click(Sender: TObject);
begin
   close;
end;

{Main Routine}
procedure TForm1.BitBtn1Click(Sender: TObject);

 type
    posXY    = array[1..2] of byte;
     b_mat    = array[1..NofP,1..NofP] of byte;
     b_vec    = array[1..NofP] of byte;
     i_vec    = array[1..NofP] of integer;
     {r_mat    = array[1..NofP,1..NofP] of reaL;}
     si_mat   = array[1..NofP,1..NofP] of ShortInt; {E}
     r_vec    = array[1..NofP] of reaL;
 var
   {=== variables to record ===}
     give_id   : b_mat;
     Oinfo     : b_mat;
     Pay_Mat   : si_mat; { payoff flow matrix }
     Pay_Vec   : r_vec; { players' payoffs }
     balance   : si_mat;
     GiveRecord : array[1..1,1..NofP] of reaL;
     SGfreq    : array[1..2,1..NofStr] of integer;
      { strategy group frequencies recorder }
     SGfreq2   : array[1..NofSG] of integer;
      { strategy group frequencies recorder }
     Gfreq     : array[1..2,1..NofSG*2] of integer;
     InGroup   : array[1..8,1..NofP] of integer;
      { InGroup id vector for each Qualification Class }
   {=== counter or identifier ===}
     converge  : boolean;  { identifier of run-convergence }
    {  LooseConv  : boolean;  } { convergence criterion is
loose or strict }
     do_repeat  : boolean;  { repeat the run or end }
     Run        : integer;  { counter of runs }
          Ite        : integer;  { counter of iterations }
     iconv      : integer;
     MaxResource : byte; { Maximum resource size for each
player }
     iSim, SSim, ESim : byte;
     iResc          : byte;
   {=== File Specifications ===}
     FNstr      : string[50];
     Dstr       : file of Agent_type;
     FNgive     : string[50];
     Dgive      : file of b_mat;

 function max_give(id : byte) : byte;
  var i  : integer;
     ss  : byte;
  begin
     ss := 2*Agent[id].max[1] + Agent[id].max[2] + 1;
     max_give := {1}ss;
  end; { of max_give }

 function min_receive(id : byte) : byte;
  var i  : integer;
     ss  : byte;
  begin
     ss := 2*Agent[id].min[1] + Agent[id].min[2] + 1;
     min_receive := {1}ss;
  end; { of min_receive }

{***** Drawing the Evolution Process per Iteration *****}
 procedure DrawProcess;
```

```
  const
     Ux = 8;
     Uy = 2;
  var
     i, j, k      : integer;
     ipage, iround : integer;
     CumCnt       : array[1..2] of integer;
     Gxy          : array[1..2,1..2] of integer;
     xy           : array[1..4] of TPoint;
  begin
     ipage  := (Run-1) div 50 + 1;
     iround := Run mod 50; if iround = 0 then iround := 50;
     if iround = 1 then begin
       with ProcessBox.Canvas do begin
         Pen.Color := clWhite; Brush.Color:= clWhite;
         Rectangle(0, 0, Width, Height);
         Pen.Color := clBlack;
         {Font.Name := 'Times New Roman';} Font.Style :=
[fsBold];
       Font.Color := clNavy; Font.Size := 11;
           TextOut(ProcessSize[1] div 2 -50, 2, 'Evolution
Process');
       Pen.Color := clBlack;
       MoveTo(10, 30); LineTo(10, ProcessSize[2]-20);
                          MoveTo(10,  ProcessSize[2]-20);
LineTo(ProcessSize[1]-10, ProcessSize[2]-20);
       Font.Color := clBlack; Font.Size := 10;
       TextOut(10,16, 'frequency');
                        TextOut(ProcessSize[1] div 2-30,
ProcessSize[2]-15, 'Generation');
       { Pen.Color := clRed;
         MoveTo(Gf[3,1,1]-5, (Gf[3,1,2]+Gf[3,2,2]) div 2);
         LineTo(Gf[3,2,1], (Gf[3,1,2]+Gf[3,2,2]) div 2);}
       end; { of with ProcessBox.Canvas }
     end; { of if }
     for i := 1 to 2 do CumCnt[i] := 0;
     for i := 1 to NofStr do
       if (Sfreq[1,i]>0) or (Sfreq[2,i]>0) then begin
         Gxy[1,1] := 10 + (iround-1)*Ux;
         Gxy[2,1] := 10 + iround*Ux;
         Gxy[1,2] := 30 + CumCnt[2]*Uy;
         Gxy[2,2] := 30 + CumCnt[1]*Uy;
         xy[1] := Point(Gxy[1,1], Gxy[1,2]);
         xy[2] := Point(Gxy[2,1], Gxy[2,2]);
              for j := 1 to 2 do CumCnt[j] := CumCnt[j] +
Sfreq[j,i];
         Gxy[1,2] := 30 + CumCnt[2]*Uy;
         Gxy[2,2] := 30 + CumCnt[1]*Uy;
         xy[4] := Point(Gxy[1,1], Gxy[1,2]);
         xy[3] := Point(Gxy[2,1], Gxy[2,2]);
         j := i;{ if i > NofSG then j := i - NofSG;}
         with ProcessBox.Canvas do begin
           Pen.Color := Scolor[j]; Brush.Color := Scolor[j];
           Polygon(xy);
         end; { of with ProcessBox.Canvas }
       end; { of if }
  end; { of DrawProcess }


 procedure CountMaxMin;
  var
       i, j, k    : integer;
       cnt        : array[1..4,1..4] of integer;

  begin
     for i := 1 to 4 do
       for j := 1 to 4 do cnt[i,j] := 0;
     for i := 1 to NofP do
       if Agent[i].ex_strategy = 4 then
       cnt[max_give(i), min_receive(i)]
         := cnt[max_give(i), min_receive(i)] + 1;
     Edit4.Text  := IntToStr(cnt[1,1]); Edit4.Refresh;
     Edit5.Text  := IntToStr(cnt[1,2]); Edit5.Refresh;
     Edit6.Text  := IntToStr(cnt[1,3]); Edit6.Refresh;
     Edit7.Text  := IntToStr(cnt[1,4]); Edit7.Refresh;
     Edit8.Text  := IntToStr(cnt[2,1]); Edit8.Refresh;
     Edit9.Text  := IntToStr(cnt[2,2]); Edit9.Refresh;
     Edit10.Text := IntToStr(cnt[2,3]); Edit10.Refresh;
     Edit11.Text := IntToStr(cnt[2,4]); Edit11.Refresh;
     Edit12.Text := IntToStr(cnt[3,1]); Edit12.Refresh;
     Edit13.Text := IntToStr(cnt[3,2]); Edit13.Refresh;
     Edit14.Text := IntToStr(cnt[3,3]); Edit14.Refresh;
     Edit15.Text := IntToStr(cnt[3,4]); Edit15.Refresh;
     Edit16.Text := IntToStr(cnt[4,1]); Edit16.Refresh;
     Edit17.Text := IntToStr(cnt[4,2]); Edit17.Refresh;
     Edit18.Text := IntToStr(cnt[4,3]); Edit18.Refresh;
     Edit19.Text := IntToStr(cnt[4,4]); Edit19.Refresh;
  end; { of CountMaxMin }

{***** Global Initialization *****}
 procedure Global_Init;
  begin
     Randomize;
     NofRuns := 500;
     SSim := 1; ESim := 10;
  end; { of Global_Init }

{***** Simulation Initialization *****}
 procedure Sim_Init;

  procedure SetInitialValues;
   var
      i, j, k   : integer;
      cnt, scnt  : integer;
      iNofRuns   : integer;
      iSimCond   : integer;
      iMutant    : integer;
      iUncertain : integer;
   begin
     for i := 1 to NofStr do Sfreq[1,i] := 0;
     for i := 1 to NofP do
                              Sfreq[1,Agent[i].ex_strategy]   :=
Sfreq[1,Agent[i].ex_strategy] + 1;
     converge := false;
     Run := 0;
     iconv := 0;
     DoMutant := true; {CheckBox1.Checked;}
   end; { of SetInitialValues }

  procedure OpenFiles;
   const
      PathName   = 'c:\R_exData\';
   var
      i, j, k    : integer;
      FName      : string[50];
      Fno        : string[2];
   begin
     case iResc of
      1: FName := 'L';
      2: FName := 'M';
      3: FName := 'S';
     end; { of case iResc of }
     case iIniD of
      1: FName := FName + '05';
      2: FName := FName + '10';
      3: FName := FName + '50';
     end; { of case iIniD of }
     Fno := IntToStr(iSim);
     if Length(Fno) < 2 then Fno := '0' + Fno;
     FNstr  := PathName + Fname + Fno + 'S.dat';
     FNgive := PathName + Fname + Fno + 'G.dat';
     AssignFile(Dstr,FNstr); Rewrite(Dstr);
     AssignFile(Dgive,FNgive); Rewrite(Dgive);
   end; { of OpenFiles }

  begin
     DefaultValues(Sender);
     SetInitialValues;
     DrawMap(Sender);
     OpenFiles;
     Edit20.Text := IntToStr(iSim);
     Edit20.Refresh;
     case iResc of
      1: Edit21.Text := '20';
```

```
        2: Edit21.Text := '10';
        3: Edit21.Text := ' 4';
       end; { of case iResc of }
       Edit21.Refresh;
       case iIniD of
        1: Edit22.Text := ' 5%';
        2: Edit22.Text := '10%';
        3: Edit22.Text := '50%';
       end; { of case iIniD of }
       Edit22.Refresh;
   end; { of Sim_Init }

{***** Recording Strategies *****}
 procedure RecordStrategies;
   var
      i, j   : integer;
   begin
      for i := 1 to NofP do write(Dstr, Agent[i]);
   end; { of RecordStrategies }

{***** Recording give_id *****}
 procedure RecordGive_Id;
   var
      i, j   : integer;
   begin
      write(Dgive, give_id);
   end; { of RecordGive_Id }

{***** Run Initialization *****}
 procedure Run_Init;
   var
      i, j   : integer;
      InSt   : string[10];
   begin
      Run := Run + 1;
      Edit1.Text := IntToStr(Run);
      Edit1.Refresh;
      Edit3.Text := 'Running';
      Edit3.Refresh;
      Ite := 0;
      for i := 1 to NofP do begin
        Pay_Vec[i] := 0.0;
        for j := 1 to NofP do begin
          balance[i,j] := 0;
          give_id[i,j] := 0;
          Oinfo[i,j] := 0;
        end; { of j loop }
      end; { of i loop }
   end; { of Run_Init }

{***** Repeat Initialization *****}
 procedure Rep_Init;
   var
      i, j   : integer;
      InSt   : string[10];
   begin
      Ite := Ite + 1;
      Edit2.Text := IntToStr(Ite);
      Edit2.Refresh;
      for i := 1 to NofP do
        for j := 1 to NofP do
          Pay_Mat[i,j] := 0;
   end; { of Rep_Init }

{***** Giving Phase *****}
 procedure Giving_Phase;
   var
      player    : integer; { player id }
      i, j, k, L  : integer;

{$I C:\Borland\Delphi6\Source\R_ex_sub1.pas}
{I c:\delphi\source\saitama.pas}
{I c:\delphi\source\tokyo.pas}
{I c:\delphi\source\tokyo2.pas}

   begin
      MaxResource := Std_Payoff[1,iResc];{Treatment}
      for player := 1 to NofP do begin
        {if player <= 50 then MaxResource := Std_Payoff[1]
                                  else MaxResource :=
Std_Payoff[2];}{Treatment}
        case Agent[player].ex_strategy of
          1: Saint(player);
          2: NonCoop(player);
          {3: Recp(player, 10);}
          4: TFT(player, 7);
        end; { of case Strategy[player] of }
      end; { of player loop }
   end; { of Giving_Phase }

{***** Account in each iteration *****}
 procedure Account;
   var
     i, j, k  : integer;
     cnt      : integer;
     ss, tt   : reaL;
     {Pinfo    : r_mat;} {E}
     tid      : array[1..4] of integer;
   begin
     for i := 1 to NofP do begin
       case Agent[i].ex_strategy of
         4: begin
              for j := 1 to NofP do
                if j <> i then balance[i,j] := - Pay_Mat[j,i];
            end; { of case 4 }
       end; { of case }
     end; { of i loop }
     {for i := 1 to NofP do
       for j := 1 to NofP do
         if i <> j then Pay_Mat[i,j] := v_f*Pay_Mat[i,j];}
     tt := 1.0;
     if Ite > 1 then
       for i := 1 to (Ite-1) do
         tt := tt * discount;
     for i := 1 to NofP do begin
       ss := 0.0;
       for j := 1 to NofP do
         if i <> j then ss := ss + v_f*Int(Pay_Mat[j,i])
         else          ss := ss +    Int(Pay_Mat[j,i]);
       Pay_Vec[i] := Pay_Vec[i] + ss * tt;
     end; { of i loop }
   end; { of Account }

{***** Run Calculation *****}
 procedure RunCalc;
   const
       Nchange   = 6; { N of strategy to be changed, upper
& lower }{Treatment}
   var
     i, j, k, L, m   : integer;
     p_id            : i_vec;
     payment         : r_vec;
     max, min, dum   : reaL;
     imax, imin, idum : integer;
     done            : boolean;
     Uid, Lid        : array[1..2, 1..Nchange] of integer;
     tieid           : i_vec;
     ntie            : integer;
     Sid, Eid        : integer;
     scnt            : integer;
     tempvec1        : dig2;
     tempvec2        : dig2;
     ClassSize, classid : integer;
     mean, ss        : reaL;
     str50           :string[50];
   begin
     {for i := 1 to NofP do
       Pay_Vec[i] := Ln(Pay_Vec[i]+1.0);}
     for i := 1 to NofP do
     if Pay_Vec[i] < 0.0 then begin
                              str50 := IntToStr(i) + ':' +
FloatToStrF(Pay_Vec[i],ffFixed, 20,3);
       form1.memo1.lines.add(str50);
     end; { of i Loop }
```

```
    ClassSize := NofP{ div 2};{Treatment}
    for classid := 1 to 1{2} do begin {Treatment}
      Sid := (Classid-1)*ClassSize + 1;
      Eid := Classid*ClassSize;
    {+++  variance judgment  +++}
      mean := 0.0; ss := 0.0;
      for i := Sid to Eid do begin
        mean := mean + Pay_Vec[i];
        ss  := ss  + Pay_Vec[i]*Pay_Vec[i];
      end; { of i Loop }
      mean := mean/Int(ClassSize);
      ss := ss - Int(ClassSize)*mean*mean;
      ss := ss / Int(ClassSize-1);
    if ss > 0.0 then begin
    {+++  Re-Ordering  +++}
      for i := 1 to NofP do begin
        p_id[i] := i;
        payment[i] := Pay_Vec[i];
      end; { of i loop }
      for i := Sid to Eid-1 do begin
        max := payment[i]; imax := p_id[i];
        for j := i to Eid do
          if payment[j] > max then begin
              dum := max; max := payment[j]; payment[j] :=
dum;
            payment[i] := max;
            idum := imax; imax := p_id[j]; p_id[j] := idum;
            p_id[i] := imax;
          end; { of if }
      end; { of i loop }
    {+++  seek upper  +++}
      max := payment[Sid+Nchange-1];
      k := Sid + Nchange - 1; done := false;
      repeat
          k := k - 1;
          if k = Sid - 1 then done := true;
          if k > Sid - 1 then
            if payment[k] > max then done := true;
      until done;
      k := k + 1;
      L := Sid + Nchange - 1; done := false;
      repeat
          L := L + 1;
          if payment[L] < max then done := true;
          if L = Eid then done := true;
      until done;
      L := L - 1; if L = Eid then L := L + 1;
      ntie := L - k + 1;
      if k > Sid then
        for i := Sid to k-1 do Uid[ClassId, i-Sid+1] := p_id[i];
      for i := k to L do tieid[i-k+1] := p_id[i];
      L := 0;
      repeat
          i := random(ntie) + 1;
          if tieid[i] <> 0 then begin
            L := L + 1;
            Uid[ClassId, k-Sid+L] := tieid[i];
            tieid[i] := 0;
          end; { of if }
      until k-Sid+L = Nchange;
    {+++  seek lower  +++}
      min := payment[Eid-Nchange+1];
      k := Eid - Nchange + 1; done := false;
      repeat
          k := k + 1;
          if k = Eid+1 then done := true;
          if k < Eid+1 then
            if payment[k] < min then done := true;
      until done;
      k := k - 1;
      L := Eid - Nchange + 1; done := false;
      repeat
          L := L - 1;
          if L = Sid - 1 then done := true; {DDDDD}
          if payment[L] > min then done := true;
      until done;
      L := L + 1;
      ntie := k - L + 1;
      if k < Eid then
          for i := Eid downto k+1 do Lid[ClassId, Eid-i+1] :=
p_id[i];
      for i := k downto L do tieid[k-i+1] := p_id[i];
      L := 0;
      repeat
          i := random(ntie) + 1;
          if tieid[i] <> 0 then begin
            L := L + 1;
            Lid[ClassId, Eid-k+L] := tieid[i];
            tieid[i] := 0;
          end; { of if }
      until Eid-k+L = Nchange;
    {+++  shuffle  +++}
      for i := 1 to Nchange do begin
        k := Uid[ClassId, i]; L := Lid[ClassId, i];
          {str50 := IntToStr(classid) + ':' + IntToStr(k) + ' ' +
IntToStr(L);
          form1.memo1.lines.add(str50);}
          j := Agent[k].ex_strategy;
          for m := 1 to 2 do begin
            tempvec1[m] := Agent[k].min[m];
            tempvec2[m] := Agent[k].max[m];
          end; { of m Loop }
          Agent[L].ex_strategy := j;
          for m := 1 to 2 do begin
            Agent[L].min[m] := tempvec1[m];
            Agent[L].max[m] := tempvec2[m];
          end; { of m Loop }
      end; { of i loop }
    end; { of if ss > 0.0 }
    end; { of clanid loop }
    {+++  Strategy Frequencies Change  +++}
      for i := 1 to NofStr do
        Sfreq[2,i] := Sfreq[1,i];
      for i := 1 to NofStr do
        Sfreq[1,i] := 0;
      for i := 1 to NofP do
                                  Sfreq[1,Agent[i].ex_strategy]  :=
Sfreq[1,Agent[i].ex_strategy] +1;
      L := 0;
      for i := 1 to NofStr do
              L := L + (Sfreq[1,i] - Sfreq[2,i])*(Sfreq[1,i] -
Sfreq[2,i]);
      { if L = 0 then
        if iconv = 0 then iconv := iconv + 1
                else converge := true;
      if (L > 0) and (iconv > 0) then iconv := 0;
      for i := 1 to NofStr do
        if Sfreq[1,i] = NofP then converge := true;}
      if Run = NofRuns then converge := true;
    end; { of RunCalc }

{*****  Producing Mutants  *****}
  procedure Mutant;
    var
      i, j, k     : integer;
      NewStrategy  : integer;
      scnt        : integer;
    begin
      if DoMutant then
      if not converge then
      for i := 1 to NofP do begin
        if random < Pmutant then begin
          NewStrategy := Agent[i].ex_strategy;
          repeat
              NewStrategy := random(NofStr) + 1;
            until (NewStrategy <> Agent[i].ex_strategy) and
(INofP[NewStrategy]>0);
          Agent[i].ex_strategy := NewStrategy;
        end; { of if random < }
        for j := 1 to 2 do
          if random < Pmutant then
            if Agent[i].min[j] = 1 then Agent[i].min[j] := 0
                              else Agent[i].min[j] := 1;
        for j := 1 to 2 do
```

```
            if random < Pmutant then
              if Agent[i].max[j] = 1 then Agent[i].max[j] := 0
                                     else Agent[i].max[j] := 1;
        end; { of i Loop }
      {+++  Strategy Frequencies Adjustment  +++}
        for i := 1 to NofStr do
          Sfreq[1,i] := 0;
        for i := 1 to NofP do
                                  Sfreq[1,Agent[i].ex_strategy]   :=
 Sfreq[1,Agent[i].ex_strategy] +1;
    end; { of Mutant }



 {*****  Drawing the iteration Results  *****}
  procedure DrawResults;
    begin
        CountMaxMin;
        DrawMap(Sender);
        DrawProcess;
    end; { of DrawResults }

 {*****  Results  *****}
  procedure Results;
 {   var
       i, j, k  : integer;
       s3       : string[3];
       FileName : string[30];
       Out      : file of r_vec;}

    begin
        DrawMap(Sender);
        Edit3.Text := 'End';
        Edit3.Refresh;
    end; { of Results }

 {*****  Close Simulation *****}
  procedure SimClose;
    begin
         CloseFile(Dstr); CloseFile(Dgive);
    end; { of SimClose }

 {*****  Finale  *****}
  procedure Finale;
    begin
          form1.memo1.lines.add('Normal End of Job');
    end; { of Finale }

 {=====  Main Routine  =====}
 begin
    Global_Init;
    for iSim := SSim to ESim do
    for iResc := 1 to 3 do
    for iIniD := 1 to 3 do begin
      Sim_Init;
      RecordStrategies;
      repeat
          Run_Init;
          if (Run mod 100) = 0 then RecordStrategies;
          repeat
              Rep_Init;
              Giving_Phase;
              Account;
          until Ite >= NofIte;
          RunCalc;
          Mutant;
          DrawResults;
          if converge then RecordGive_Id;
      until converge;
      Results;
      SimClose;
    end; { of iIniD }
    Finale;
 end;

 end.
```

協力呼びかけモデル

```
{ Social Gathering Simulation                    }
{              Written in Borland Delphi Ver.6 J      }
{     Recording version                  'SG_r1'   }
{  Eiji TAKAGI   Prog. Start:2002.02.16              }
unit SG_r1a;

interface

uses
    Windows, Messages, SysUtils, Classes, Graphics,
Controls, Forms, Dialogs,
 ExtCtrls, StdCtrls, Buttons, Menus, jpeg;

type
 TForm1 = class(TForm)
   InfoBox: TPaintBox;
   MapBox: TPaintBox;
   ProcessBox: TPaintBox;
   EventBox: TPaintBox;
   Memo1: TMemo;
   Panel1: TPanel;
   BitBtn1: TBitBtn;
   BitBtn2: TBitBtn;
   CheckBox1: TCheckBox;
   RadioGroup1: TRadioGroup;
   RadioGroup2: TRadioGroup;
   RadioGroup3: TRadioGroup;
   RadioGroup4: TRadioGroup;
   Label1: TLabel;
   Label2: TLabel;
   Label3: TLabel;
   StaticText1: TStaticText;
   StaticText2: TStaticText;
   StaticText3: TStaticText;
   Image1: TImage;
   Edit1: TEdit;
   Edit2: TEdit;
   Edit3: TEdit;
   Edit4: TEdit;
   Edit5: TEdit;
   Edit6: TEdit;
   Edit7: TEdit;
   Edit8: TEdit;
   Edit9: TEdit;
   Edit10: TEdit;
   Edit11: TEdit;
   Edit12: TEdit;
   Edit13: TEdit;
   Edit14: TEdit;
   Edit15: TEdit;
   Edit16: TEdit;
   Edit17: TEdit;
   Edit18: TEdit;
   Edit19: TEdit;
   Label4: TLabel;
   Label5: TLabel;
   Label6: TLabel;
   Label7: TLabel;
   Label8: TLabel;
   Label9: TLabel;
   Label10: TLabel;
   Label11: TLabel;
   Label12: TLabel;
   Label13: TLabel;
   CheckBox2: TCheckBox;
   Label14: TLabel;
   Label15: TLabel;
   procedure DefaultValues(Sender: TObject);
   procedure PaintB(Sender: TObject);
   procedure PaintB1(Sender: TObject);
   procedure InfoFrame(Sender: TObject);
   procedure AgentMap(Sender: TObject);
   procedure DrawMap(Sender: TObject);
   procedure ProcessFrame(Sender: TObject);
   procedure EventFrame(Sender: TObject);
   procedure FormCreate(Sender: TObject);
   procedure BitBtn1Click(Sender: TObject);
   procedure BitBtn2Click(Sender: TObject);
{  procedure RadioGroup1Click(Sender: TObject);
   procedure RadioGroup2Click(Sender: TObject);
   procedure Button1Click(Sender: TObject);}
 private
  { Private 宣言 }
 public
  { Public 宣言 }
 end;

const
     MapSize    : array[1..2] of integer = (250, 250);
     ProcessSize : array[1..2] of integer = (400, 250);
     EventSize   : array[1..2] of integer = (400, 250);
     InfoBoxSize : array[1..2] of integer = ( 5,  5);
     Gf         : array[1..3, 1..2, 1..2] of integer
                  = ((( 50,100), (450, 500)), ((550,100), (750,
300)),
                  ((550,400), (950,600)));
     NAgents    = 100; { N of players }
       {NofRounds   = 200}{300;} { Number of Rounds for
each run }
     Pmutant    = 0.015;  { Pr of Mutant Emergence }
     discount   = 0.95; { discount factor weight }
     AgentColor  : array[0..4] of TColor =
                  (clGray, clYellow, clFuchsia, clAqua, clRed);
                          {Back, Manager, DefectManager,
Cooperator, Defector}
     SColor     : array[1..4] of TColor =
                  (clRed, clOlive, clLime, clAqua);
     EventColor  : array[1..4] of TColor =
                  (clRed, clBlue, clLime, clFuchsia);
     CellColor   : array[0..5] of TColor =
                   (clBlack, clGray, clSilver, clWhite, clPurple,
clOlive);
     CellStyle   : array[0..5] of TBrushStyle =
                  (bsClear, bsVertical, bsCross, bsDiagCross,
bsSolid, bsBDiagonal);
 type
     dig2     = array[1..2]  of byte;
     dig3     = array[1..3]  of byte;
     dig5     = array[1..5]  of byte;
     dig19    = array[1..19] of byte;
     word_vec   = array[1..NAgents] of word;

     Agent_type = record
                  state      : ShortInt;
                          {0:none, 1: manager, 2:cooperator,
3:defector}
                  profit     : reaL;
                  contact    : word_vec;
                  defect     : word_vec;
                  strategy   :dig19;
               end;
{Shortint  -128..127 符号付き 8 ビット
Smallint   -32768..32767       符号付き 16 ビット
Longint    -2147483648..2147483647     符号付き 32 ビッ
ト
Int64      -2^63..2^63-1       符号付き 64 ビット
Byte       0..255     符号なし 8 ビット
Word       0..65535  符号なし 16 ビット
Longword 0..4294967295       符号なし 32 ビット}
var
     Form1      : TForm1;
   {=== variables first determined ===}
     Agent      : array[1..NAgents] of Agent_type;
   {=== Important variables ===}
     {Sfreq     : array[1..2, 1..NofStr] of integer;}
      { strategy frequencies recorder }
   {=== Condition Variables ===}
     Std_Payoff  : array[1..2] of byte; { Standard payoffs }

implementation
```

```
{$R *.DFM}

 function N_invite(id : byte) : byte;
  const
     n_dim          = 4;
  var
     i, j          : integer;
     inc           : integer;
     vector        : array[1..n_dim] of byte;
  begin
     for i := 1 to n_dim do vector[i] := Agent[id].strategy[i];
     j := 0; inc := 1;
     for i := 1 to n_dim do begin
       if vector[n_dim+1-i] = 1 then j := j + inc;
       inc := inc *2;
     end; { of i loop }
     N_invite := j;
  end; { of N_invite }

 function apply_s_exploit(id : byte) : boolean;
  const
     n_dim          = 1;
  var
     i, j, k       : integer;
     inc           : integer;
     vector        : array[1..n_dim] of byte;
     bx            : boolean;
  begin
                    for i := 1 to n_dim do vector[i] :=
Agent[id].strategy[i+4];
     if vector[1] = 1 then bx := true
                  else bx := false;
     apply_s_exploit := bx;
  end; { of apply_s_exploit }

 function s_exploit(id : byte) : real;
  const
     n_dim          = 2;
  var
     i, j, k       : integer;
     inc           : integer;
     vector        : array[1..n_dim] of byte;
  begin
                    for i := 1 to n_dim do vector[i] :=
Agent[id].strategy[i+5];
     j := 0; inc := 1;
     for i := 1 to n_dim do begin
       if vector[n_dim+1-i] = 1 then j := j + inc;
       inc := inc *2;
     end; { of i loop }
     k := 1;
     for i := 1 to n_dim do k := k * 2;
     s_exploit := Int(j)/Int(k-1);
  end; { of s_exploit }

 function apply_s_risk(id : byte) : boolean;
  const
     n_dim          = 1;
  var
     i, j, k       : integer;
     inc           : integer;
     vector        : array[1..n_dim] of byte;
     bx            : boolean;
  begin
                    for i := 1 to n_dim do vector[i] :=
Agent[id].strategy[i+7];
     if vector[1] = 1 then bx := true
                  else bx := false;
     apply_s_risk := bx;
  end; { of apply_s_risk }

 function s_risk(id : byte) : real;
  const
     n_dim          = 2;
  var
     i, j, k       : integer;
     inc           : integer;
     vector        : array[1..n_dim] of byte;
  begin
                    for i := 1 to n_dim do vector[i] :=
Agent[id].strategy[i+8];
     j := 0; inc := 1;
     for i := 1 to n_dim do begin
       if vector[n_dim+1-i] = 1 then j := j + inc;
       inc := inc *2;
     end; { of i loop }
     k := 1;
     for i := 1 to n_dim do k := k * 2;
     s_risk := Int(j)/Int(k-1);
  end; { of s_risk }

 function p_accept(id : byte) : real;
  const
     n_dim          = 2;
  var
     i, j, k       : integer;
     inc           : integer;
     vector        : array[1..n_dim] of byte;
  begin
                    for i := 1 to n_dim do vector[i] :=
Agent[id].strategy[i+10];
     j := 0; inc := 1;
     for i := 1 to n_dim do begin
       if vector[n_dim+1-i] = 1 then j := j + inc;
       inc := inc *2;
     end; { of i loop }
     k := 1;
     for i := 1 to n_dim do k := k * 2;
     p_accept := Int(j)/Int(k-1);
  end; { of p_accept }

 function p_defect(id : byte) : real;
  const
     n_dim          = 3;
  var
     i, j, k       : integer;
     inc           : integer;
     vector        : array[1..n_dim] of byte;
  begin
                    for i := 1 to n_dim do vector[i] :=
Agent[id].strategy[i+12];
     j := 0; inc := 1;
     for i := 1 to n_dim do begin
       if vector[n_dim+1-i] = 1 then j := j + inc;
       inc := inc *2;
     end; { of i loop }
     k := 1;
     for i := 1 to n_dim do k := k * 2;
     p_defect := Int(j)/Int(k-1);
  end; { of p_defect }

 function give_bribe(id : byte) : boolean;
  const
     n_dim          = 1;
  var
     i, j, k       : integer;
     inc           : integer;
     vector        : array[1..n_dim] of byte;
     bx            : boolean;
  begin
                    for i := 1 to n_dim do vector[i] :=
Agent[id].strategy[i+15];
     if vector[1] = 1 then bx := true
                  else bx := false;
     give_bribe := bx;
  end; { of give_bribe }

 function take_bribe(id : byte) : boolean;
  const
     n_dim          = 1;
  var
     i, j, k       : integer;
     inc           : integer;
     vector        : array[1..n_dim] of byte;
```

```
      bx          : boolean;
  begin
                  for i := 1 to n_dim do vector[i] :=
Agent[id].strategy[i+16];
      if vector[1] = 1 then bx := true
                  else bx := false;
      take_bribe := bx;
  end; { of take_bribe }

 function trust(id : byte) : reaL;
  const
     n_dim        = 2;
  var
     i, j, k      : integer;
     inc          : integer;
     vector       : array[1..n_dim] of byte;
  begin
                  for i := 1 to n_dim do vector[i] :=
Agent[id].strategy[i+17];
     j := 0; inc := 1;
     for i := 1 to n_dim do begin
       if vector[n_dim+1-i] = 1 then j := j + inc;
       inc := inc *2;
     end; { of i loop }
     k := 1;
     for i := 1 to n_dim do k := k * 2;
     trust := Int(j)/Int(k-1);
  end; { of trust }

{*****  Random Number Generator: Normal Distribution
*****}
 function Normal(m, s: real) : real;
{ inputs : m(mean) & s(standard deviation)          }
  var
     u, v   : array[1..2] of real;
     w, y   : real;
     i      : integer;
  begin
     w := 0.0;
     repeat
        for i := 1 to 2 do begin
          u[i] := random;
          v[i] := 2.0 * u[i] - 1.0;
        end; { of i loop }
        w := v[1]*v[1] + v[2]*v[2];
     until w <= 1.0;
     y := Sqrt(-2.0*Ln(w)/w);
     for i := 1 to 2 do
       u[i] := v[i]*y;
     Normal := u[1]*s + m;
  end; { of Normal }

procedure TForm1.DefaultValues(Sender: TObject);
   var
      i, j, k    : integer;
      cnt, scnt  : integer;
      iNofGens   : integer;
      iSimCond   : integer;
      iUncertain : integer;

   begin
      iSimCond := RadioGroup1.ItemIndex + 1;
      { iNofGens := RadioGroup2.ItemIndex + 1;
      case iNofGens of
       1 : NofGenerations :=  50;
       2 : NofGenerations := 100;
       3 : NofGenerations := 200;
       4 : NofGenerations := 500;
      end;} { of case iNofGens }
      {case RadioGroup3.ItemIndex of
       0 : Std_Payoff[1] := 20;
       1 : Std_Payoff[1] := 10;
       2 : Std_Payoff[1] :=  4;
      end;} { of case iUncertain }
      {case RadioGroup4.ItemIndex of
       0 : Std_Payoff[2] := 20;
       1 : Std_Payoff[2] := 10;
       2 : Std_Payoff[2] :=  4;
      end;} { of case iUncertain } {Treatment}
   end; { of DefaultValues }

procedure TForm1.AgentMap(Sender: TObject);
  var
     i, j, k  : integer;
     InSt     : string[15];
  begin
     with MapBox.Canvas do begin
       Pen.Color := clBlack; Brush.Color:= clWhite;
       Rectangle(0, 0, Width, Height);
        Font.Color := clNavy; Font.Size := 12; Font.Style :=
[fsBold];
       TextOut(70{Width div 2 - 100}, 3, 'Agent Map');
     end; { of with MapBox.Canvas }
  end; { of AgentMap }

{*****  Drawing Strategy Map & Displaying Strategy
Frequencies *****}
 procedure TForm1.DrawMap(Sender: TObject);
  var
     i, j, k   : integer;

  procedure DrawPlayer(pL : integer);
   var
     i, j, k   : integer;
     Gx, Gy    : integer;
     Sid       : integer;
   begin
     i := pL mod 10;
     if i = 0 then i := 10;
     Gx := 25 + 20*(i-1);
     j := (pL-1) div 10 + 1;
     Gy := 25 + 20*(j-1);
     Sid := Agent[pL].state;
     with MapBox.Canvas do begin
       Pen.Color := clNavy; Font.Color := AgentColor[Sid];
Font.Size := 8;
            Brush.Color := AgentColor[Sid]; Font.Style :=
[fsBold];
       Rectangle(Gx, Gy, Gx+20, Gy+20);
       {TextOut(Gx+5, Gy+2, Smark[Sid]);}
     end; { of with MapBox.Canvas }
   end; { of DrawPlayer }

  begin
    for i := 1 to NAgents do DrawPlayer(i);
  end; { of DrawMap }

  procedure TForm1.InfoFrame(Sender: TObject);
   var
     i, j, k  : integer;
     InSt     : string[15];
     n1       : integer;
   begin
   end; { of InfoFrame }

  procedure TForm1.ProcessFrame(Sender: TObject);
   begin
     with ProcessBox.Canvas do begin
       Pen.Color := clBlack; Brush.Color:= clWhite;
       Rectangle(0, 0, Width, Height);
         {Font.Name := 'Times New Roman';} Font.Style :=
[fsBold];
       Font.Color := clNavy; Font.Size := 11;
           TextOut(ProcessSize[1] div 2 -50, 2, 'Evolution
Process');
       Pen.Color := clBlack;
       MoveTo(10, 30); LineTo(10, ProcessSize[2]-20);
                        MoveTo(10,  ProcessSize[2]-20);
LineTo(ProcessSize[1]-10, ProcessSize[2]-20);
       Font.Color := clBlack; Font.Size := 10;
       TextOut(10,16, 'frequency');
                        TextOut(ProcessSize[1] div 2-30,
ProcessSize[2]-15, 'Generation');
     end; { of with ProcessBox.Canvas }
```

```
    end; { of ProcessFrame }

  procedure TForm1.EventFrame(Sender: TObject);
   begin
      with EventBox.Canvas do begin
        Pen.Color := clBlack; Brush.Color:= clWhite;
        Rectangle(0, 0, Width, Height);
          {Font.Name := 'Times New Roman';} Font.Style :=
[fsBold];
        Font.Color := clNavy; Font.Size := 11;
        TextOut(EventSize[1] div 2 -50, 2, 'Event Types');
        Pen.Color := clBlack;
        MoveTo(10, 30); LineTo(10, EventSize[2]-20);
                              MoveTo(10,   EventSize[2]-20);
LineTo(EventSize[1]-10, EventSize[2]-20);
        Font.Color := clBlack; Font.Size := 10;
        TextOut(10,16, 'frequency');
            TextOut(EventSize[1] div 2-30, EventSize[2]-15,
'Generation');
      end; { of with EventBox.Canvas }
   end; { of EventFrame }

procedure TForm1.PaintB(Sender: TObject);
 begin
       InfoBox.Width := InfoBoxSize[1]; InfoBox.Height :=
InfoBoxSize[2];
    with InfoBox.Canvas do begin
      Brush.Color := clWindow;
      Rectangle(0, 0, Width, Height);
      Pen.Color := clWindow;
    end; { of with InfoBox }
      MapBox.Width  := MapSize[1];  MapBox.Height  :=
MapSize[2];
    with MapBox.Canvas do begin
      Brush.Color := clWindow;
      Rectangle(0, 0, Width, Height);
      Pen.Color := clWindow;
    end; { of with MapBox }
    ProcessBox.Width  := ProcessSize[1];
    ProcessBox.Height := ProcessSize[2];
    with ProcessBox.Canvas do begin
      Brush.Color := clWindow;
      Rectangle(0, 0, Width, Height);
      Pen.Color := clWindow;
    end; { of with ProcessBox }
    EventBox.Width  := EventSize[1];
    EventBox.Height := EventSize[2];
    with EventBox.Canvas do begin
      Brush.Color := clWindow;
      Rectangle(0, 0, Width, Height);
      Pen.Color := clWindow;
    end; { of with EventBox }
 end; { of PaintB }

procedure TForm1.PaintB1(Sender: TObject);
 begin
       InfoBox.Width := InfoBoxSize[1]; InfoBox.Height :=
InfoBoxSize[2];
    with InfoBox.Canvas do begin
      Brush.Color := clWindow;
      Rectangle(0, 0, Width, Height);
      Pen.Color := clWindow;
        {Font.Name := 'Times New Roman'; Font.Style :=
[fsBold];}
    end; { of with InfoBox }
      MapBox.Width  := MapSize[1];  MapBox.Height  :=
MapSize[2];
    with MapBox.Canvas do begin
      Brush.Color := clWindow;
      Rectangle(0, 0, Width, Height);
      Pen.Color := clWindow;
    end; { of with MapBox }
    ProcessBox.Width  := ProcessSize[1];
    ProcessBox.Height := ProcessSize[2];
    with ProcessBox.Canvas do begin
      Brush.Color := clWindow;
      Rectangle(0, 0, Width, Height);
      Pen.Color := clWindow;
    end; { of with ProcessBox }
    EventBox.Width  := EventSize[1];
    EventBox.Height := EventSize[2];
    with EventBox.Canvas do begin
      Brush.Color := clWindow;
      Rectangle(0, 0, Width, Height);
      Pen.Color := clWindow;
    end; { of with EventBox }
    DefaultValues(Sender);
    AgentMap(Sender);
    ProcessFrame(Sender);
    InfoFrame(Sender);
    DrawMap(Sender);
    EventFrame(Sender);
 end; { of PaintB1 }

procedure TForm1.FormCreate(Sender: TObject);
begin
   PaintB1(Sender);
end;

{
procedure TForm1.RadioGroup1Click(Sender: TObject);
begin
   PaintB(Sender);
   InfoFrame(Sender);
   AgentMap(Sender);
   ProcessFrame(Sender);
end;

procedure TForm1.RadioGroup2Click(Sender: TObject);
begin
   PaintB(Sender);
   InfoFrame(Sender);
   AgentMap(Sender);
   ProcessFrame(Sender);
end;
}

procedure TForm1.BitBtn2Click(Sender: TObject);
begin
   close;
end;

{Main Routine}
procedure TForm1.BitBtn1Click(Sender: TObject);

 type
    posXY    = array[1..2] of byte;
    {b_mat    = array[1..NAgents,1..NAgents] of byte;}
    b_vec    = array[1..NAgents] of byte;
    {i_vec    = array[1..NAgents] of integer;}
    {r_mat    = array[1..NAgents,1..NAgents] of reaL;}
    {si_mat   = array[1..NAgents,1..NAgents] of ShortInt;}
    r_vec    = array[1..NAgents] of reaL;
    AgentRec  = record
                  profit    : reaL;
                  strategy  :dig19;
                end;
    SumRec    = record
                  stypes   : array[1..4] of integer;
                  msize    : reaL;
                  Coop     : reaL;
                  mtrust   : reaL;
                  m_se     : reaL;
                  m_sr     : reaL;
                end;

 const
    SSim     = 1;
    ESim     = 10;
 var
   {=== variables to record ===}
    {give_id    : b_mat;
    Oinfo      : b_mat;
    Pay_Mat    : si_mat;} { payoff flow matrix }
```

```
    {Pay_Vec    : r_vec;} { players' payoffs }
    {balance    : si_mat;
    GiveRecord  : array[1..1,1..NAgents] of reaL;
    InGroup     : array[1..8,1..NAgents] of integer;}
     { InGroup id vector for each Qualification Class }
  {=== Variables to be determined first  ===}
    NofGenerations : integer;  { Number of Generations per
simulation }
    NofRounds     : integer;
    Wtemptation   : reaL;
    DoMutant      : boolean; { Introduce Mutants }
    bribe         : boolean;
  {=== Variables for Graphs  ===}
    Gstypes    : array[1..2,1..4] of integer;
    GmSize     : array[1..2] of reaL;
    GCoop      : array[1..2] of reaL;
    GmTrust    : array[1..2] of reaL;
    Gm_se      : array[1..2] of reaL;
    Gm_sr      : array[1..2] of reaL;
    countNC    : array[1..2] of integer;
  {=== counter or identifier ===}
    converge   : boolean;   { identifier of run-convergence }
   {  LooseConv   : boolean;  } { convergence criterion is
loose or strict }
    do_repeat  : boolean;   { repeat the run or end }
    iCond      : integer;   { counter of conditions }
    iSim       : integer;   { counter of simulations }
    iGeneration : integer;   { counter of Generations }
        iRound     : integer;   { counter of Rounds }
    iconv      : integer;
    MaxResource : byte; { Maximum resource size for each
player }
    ActOrder   : array[1..NAgents] of byte;
  {=== File Specifications ===}
    FN_Agent   : string[50];
    Out_Agent  : file of AgentRec;
    FN_summary : string[50];
    Out_summary : file of SumRec;


  procedure MakeActOrder;
   var
    i, j, k   : integer;
    seL1, seL2 : integer;
   begin
     for i := 1 to NAgents do ActOrder[i] := i;
     for i := 1 to 10 do
       for j := 1 to NAgents do begin
                 seL1  :=  random(NAgents)+1;  seL2  :=
random(NAgents)+1;
         k := ActOrder[seL1];
         ActOrder[seL1] := ActOrder[seL2];
         ActOrder[seL2] := k;
       end; { of j Loop }
   end; { of MakeActOrder }

{***** Global Initialization *****}
 procedure Global_Init;
  begin
    Randomize;
    DefaultValues(Sender);
  end; { of Global_Init }

{***** Simulation Initialization *****}
 procedure Sim_Init;

  procedure SetInitialValues;
   var
     i, j, k   : integer;
     cnt, scnt  : integer;
     iNofGens   : integer;
     iSimCond   : integer;
     iMutant    : integer;
     iUncertain : integer;
   begin
     case RadioGroup1.ItemIndex of
      0 : NofGenerations :=  50;
      1 : NofGenerations := 100;
      2 : NofGenerations := 200;
      3 : NofGenerations := 500;
     end; { of case RadioGroup1.ItemIndex }
     case RadioGroup2.ItemIndex of
      0 : NofRounds := 100;
      1 : NofRounds := 200;
     end; { of case RadioGroup2.ItemIndex }
     case iCond {RadioGroup3.ItemIndex} of
      0 : Wtemptation :=  0.1;
      1 : Wtemptation :=  2.0/3.0;
      2 : Wtemptation :=  1.0;
      3 : Wtemptation :=  5.0/3.0;
     end; { of case RadioGroup3.ItemIndex }
     converge := false;
     iGeneration := 0;
     iconv := 0;
     DoMutant := CheckBox1.Checked;
     bribe    := CheckBox2.Checked;
     for i := 1 to NAgents do begin
       Agent[i].state  := 0;
       Agent[i].profit := 0.0;
       for j := 1 to NAgents do begin
         Agent[i].contact[j] := 0;
         Agent[i].defect[j]  := 0;
       end; { of j Loop }
       for j :=  1 to 19 do Agent[i].strategy[j] := random(2);
            {for j :=  5 to  5 do Agent[i].strategy[j] := 1;}
{NNNNN}
            {for j :=  8 to  8 do Agent[i].strategy[j] := 1;}
{NNNNN}
       if not bribe then
            for j := 16 to 17 do Agent[i].strategy[j] := 0;
{NNNNN}
     end; { of i Loop }
   end; { of SetInitialValues }

  procedure InitialIndex;
   var
     i, pid  : integer;
   begin
     for i := 1 to 4 do Gstypes[1,i] := 0;
     for pid := 1 to NAgents do begin
       i := 1;
       if apply_s_risk(pid) then i := i + 1;
       if apply_s_exploit(pid) then i := i + 2;
       Gstypes[1,i] := Gstypes[1,i] + 1;
     end; { of pid Loop }
   end; { of InitialIndex }

  procedure OpenFiles;
   const
     PathName   = 'c:\SGData\';
   var
     i, j, k    : integer;
     FName      : string[50];
     Fno        : string[2];
   begin
     FName := '';
      if CheckBox2.checked then FName := Fname + 'B'
{with Bribe }
                    else FName := Fname + 'N'; {without
Bribe }
     FName := FName + IntToStr(iCond);
     Fno := IntToStr(iSim);
     if Length(Fno) < 2 then Fno := '0' + Fno;
     FN_Agent  := PathName + Fname + Fno + 'A.dat';
     FN_Summary := PathName + Fname + Fno + 'S.dat';
                    AssignFile(Out_Agent,FN_Agent);
Rewrite(Out_Agent);
                AssignFile(Out_Summary,FN_Summary);
Rewrite(Out_Summary);
   end; { of OpenFiles }

  begin
    Edit10.Text := IntToStr(iCond); Edit10.Refresh;
    Edit11.Text := IntToStr(iSim);  Edit11.Refresh;
```

```
      {DefaultValues(Sender);}
      SetInitialValues;
      DrawMap(Sender);
      InitialIndex;
      OpenFiles;
    end; { of Sim_Init }

{*****  Generation Initialization  *****}
  procedure Generation_Init;
    var
      i, j    : integer;
      InSt    : string[10];

    begin
      iGeneration  := iGeneration + 1;
      Edit1.Text := IntToStr(iGeneration);
      Edit1.Refresh;
      {Edit3.Text := 'Running';
      Edit3.Refresh;}
      iRound := 0;
      for i := 1 to NAgents do begin
        Agent[i].state  := 0;
        Agent[i].profit := 0.0;
        for j := 1 to NAgents do begin
          Agent[i].contact[j] := 0;
          Agent[i].defect[j]  := 0;
        end; { of j Loop }
      end; { of i Loop }
      for i := 1 to 2 do countNC[i] := 0;
    end; { of Generation_Init }

{*****  Round Initialization  *****}
  procedure Round_Init;
    var
      i, j    : integer;
      InSt    : string[10];
    begin
      iRound := iRound + 1;
      Edit2.Text := IntToStr(iRound);
      Edit2.Refresh;
      MakeActOrder;
      {for i := 1 to NAgents do
        for j := 1 to NAgents do
          Pay_Mat[i,j] := 0;}
    end; { of Round_Init }

{*****  Gathering Phase  *****}
  procedure Gathering_Phase;
    var
      iAgent  : integer;
      Aid     : integer;
      i, j    : integer;
      a_act   : b_vec;
      {values  0:none, 1:manager, 2:briber, 3: included }
      Nbribers: byte;

    procedure BribeTransaction(manager : byte);
      var
        i, j, k    : integer;
        n, m       : integer;
        candidates : b_vec;
        bribers    : b_vec;
      begin
        Nbribers := 0;
        if bribe then begin
          for i := 1 to NAgents do begin
            candidates[i] := 0;
            bribers[i] := 0;
          end; { of i Loop }
        {Rumor}
          n := 0;
          repeat
            i := random(NAgents) + 1;
            if i <> manager then
              if candidates[i] = 0 then
              begin
                n := n + 1;
                candidates[n] := i;
              end; { of if }
          until n >= 20;
        {Bribe Offer}
          m := 0;
          for i := 1 to n do
            if give_bribe(candidates[i]) then
            if random < P_defect(candidates[i]) then begin
              m := m + 1;
              bribers[m] := candidates[i];
            end; { of if random < }
        {Bribe Accept}
          if N_invite(manager) div 2 > 0 then
          if (m > 0) and take_bribe(manager) then begin
            if m > N_invite(manager) div 2 then
              repeat
                i := random(m) + 1;
                if i < m then
                  for j := i to m-1 do
                    bribers[i] := bribers[i+1];
                bribers[m] := 0;
                m := m - 1;
              until m <= N_invite(manager) div 2;
            for i := 1 to m do
              a_act[bribers[i]] := 2;
            Nbribers := m;
          end; { of if (m > 0) and take_bribe(manager) }
        end; { of if bribe }
      end; { of BribeTransaction }

    procedure Invitation(manager : byte);
      var
        i, j, k    : integer;
        n, m       : integer;
        candidates : b_vec;
        defect_rate : real;
        safe_rate  : real;
        pid        : byte;
      begin
        n := N_invite(manager) - Nbribers;
        if n > 0 then begin
        {===  Invitation  ===}
          if (iRound <= 0) or (not apply_s_exploit(manager))
then {NNNNN}
          {Invitation is random-based. }
          begin
            m := 0;
            repeat
              i := random(NAgents) + 1;
              if a_act[i] = 0 then begin
                a_act[i] := 3;
                m := m + 1;
              end; { of if }
            until m = n;
          end
          else
          {Invitation is exploit-rate based. }
          begin
            for i := 1 to NAgents do candidates[i] := 0;
            m := 0;
            for i := 1 to NAgents do
              if a_act[i] = 0 then begin
                if Agent[manager].contact[i] > 0 then
                  defect_rate := Int(Agent[manager].defect[i])/
                                 Int(Agent[manager].contact[i])
                else
                  defect_rate := trust(manager);
                    if defect_rate <= s_exploit(manager) then
begin
                  m := m + 1;
                  candidates[m] := i;
                end; { of if }
              end; { of if a_act[i] = 0 }
            if m > n then
              repeat
                i := random(m) + 1;
                for j := i to m-1 do
```

```
                candidates[j] := candidates[j+1];
              m := m - 1;
            until m = n;
          if m > 0 then
              for i := 1 to m do a_act[candidates[i]] :=
3;{NNNNN}

      end; {of if (iRound <= 2) or not apply_e_exploit }
    {=== Response to Invitation ===}
      for i := 1 to NAgents do
      if a_act[i] = 3 then begin
        if apply_s_risk(i) then
        {Response is risk-based. }
        begin
          safe_rate := 1.0;
          for j := 1 to NAgents do
            if (j<>i) and (a_act[j]>0) then begin
              if Agent[i].contact[j] > 0 then
                defect_rate := Int(Agent[i].defect[j])/
                        Int(Agent[i].contact[j])
              else
                defect_rate := trust(i);
              safe_rate := safe_rate * (1.0 - defect_rate);
            end; { of if (j<>i) and (a_act[j]>0) }
          if (1.0 - safe_rate) > s_risk(i) then a_act[i] := 0;
        end
        else
        {Response is random-based. }
        begin
          if random >= p_accept(i) then a_act[i] := 0;
        end; { of if apply_s_risk(i) }
      end; { of if a_act[i] = 3 }
    end; {of if n > 0 }
  end; { of Invitation }

  procedure Gathering(manager : byte);
    var
      i, j, k    : integer;
      pid        : byte;
      Nt, Nc, Nd  : byte;
      n, m       : integer;


      candidates  : b_vec;
      defect_rate : reaL;
      safe_rate   : reaL;

    function StandardValue(n: byte): reaL;
      begin
        StandardValue := 3.0*Sqrt(Int(n));
      end; { of StandardValue }

    begin
      for pid := 1 to NAgents do begin
        case a_act[pid] of
          0: Agent[pid].state := 0;
          1: if random >= p_defect(pid) then
                Agent[pid].state := 1
             else
                Agent[pid].state := 2;
          2: Agent[pid].state := 4;
          3: if random >= p_defect(pid) then
                Agent[pid].state := 3
             else
                Agent[pid].state := 4;
        end; { of case Agent[pid].state of }
      end; { of pid Loop }
      Nt := 0; Nc := 0; Nd := 0;
      for pid := 1 to NAgents do begin
        if (Agent[pid].state = 1) or (Agent[pid].state = 3)
          then Nc := Nc + 1;
        if (Agent[pid].state = 2) or (Agent[pid].state = 4)
          then Nd := Nd + 1;
      end; { of pid Loop }
      Nt := Nc + Nd;
      if Nt = 1 then begin
        Nc := 1; Nd := 0;
        Agent[manager].state := 1;
      end; { of if }
      countNC[1] := countNC[1] + Nt;
      countNC[2] := countNC[2] + Nc;
    {Payoffs}
      for pid := 1 to NAgents do begin
        if (Agent[pid].state = 1) or (Agent[pid].state = 2)
          then Agent[pid].profit := Agent[pid].profit
               + StandardValue(Nt) - 0.5*Int(Nt);
        if (Agent[pid].state = 3) or (Agent[pid].state = 4)
          then Agent[pid].profit := Agent[pid].profit
               + StandardValue(Nt)/10.0;
      end; { of pid Loop }
      if Nd > 0 then
      for pid := 1 to NAgents do begin
        if (Agent[pid].state >= 1)
          then Agent[pid].profit := Agent[pid].profit
               - Int(Nd)*StandardValue(Nt)/5.0;
        if (Agent[pid].state = 2) or (Agent[pid].state = 4)
          then Agent[pid].profit := Agent[pid].profit
                                                          +
Wtemptation*Int(Nt-1)*StandardValue(Nt)/10.0; {NNNNN}
      end; { of pid Loop }
      if Nbribers > 0 then
      for pid := 1 to NAgents do
        if a_act[pid] = 2 then begin
          Agent[pid].profit := Agent[pid].profit
               - Wtemptation*3.0*StandardValue(Nt)/10.0;
          Agent[manager].profit := Agent[manager].profit
               + Wtemptation*3.0*StandardValue(Nt)/10.0;
        end; { of if }
    {Memory-encoding}
      for pid := 1 to NAgents do
        if Agent[pid].state > 0 then begin {NNNNN}
          for i := 1 to NAgents do begin
            if (Agent[i].state = 1) or (Agent[i].state = 3) then
begin
              Agent[pid].contact[i] := Agent[pid].contact[i] +
1;
            end; { of if }
            if (Agent[i].state = 2) or (Agent[i].state = 4) then
begin
              Agent[pid].contact[i] := Agent[pid].contact[i] +
1;
              Agent[pid].defect[i]  := Agent[pid].defect[i]  +
1;
            end; { of if }
            {if give_bribe(pid) and (i=manager) and (Nd>0)
then
              Agent[pid].defect[i]  := Agent[pid].defect[i]  +
1;}{NNNNN}
          end; { of i Loop }
        end; { of if Ageny[pid].state > 0 }{NNNNN}
    end; { of Gathering }

  begin
    for iAgent := 1 to NAgents do begin
      {Edit3.Text := IntToStr(iAgent); Edit3.Refresh;}
      for i := 1 to NAgents do begin
        a_act[i] := 0; Agent[i].state := 0;
      end; { of i Loop }
      Aid := ActOrder[iAgent];
      a_act[Aid] := 1;
      {Edit4.Text := 'Act'; Edit4.Refresh;}
      BribeTransaction(Aid);
      {Edit4.Text := 'Bribe'; Edit4.Refresh;}
      Invitation(Aid);
      {Edit4.Text := 'Invite'; Edit4.Refresh;}
      Gathering(Aid);
      {Edit4.Text := 'Gather'; Edit4.Refresh;}
      DrawMap(Sender);
      {if iGeneration = 25 then}      {NNNNN}
        {for i := 1 to 30000 do
          for j := 1 to 30000 do begin
          end;}
    end; { of iAgent Loop }
  end; { of Gathering_Phase }
```

```
{***** Calculationg Index *****}
 procedure CalcIndex;
   {=== Variables for Graphs ===}
     {Gstypes    : array[1..2,1..4] of integer;
      GmSize     : array[1..2] of reaL;
      GCoop      : array[1..2] of reaL;
      GmTrust    : array[1..2] of reaL;
      Gm_se      : array[1..2] of reaL;
      Gm_sr      : array[1..2] of reaL;
      countNC    : array[1..2] of integer;}
   var
      i, j, k   : integer;
      pid       : byte;
   begin
      for i := 1 to 4 do Gstypes[2,i] := Gstypes[1,i];
      GmSize[2] := GmSize[1]; GCoop[2] := GCoop[1];
      GmTrust[2] := GmTrust[1];
      Gm_se[2]  := Gm_se[1];  Gm_sr[2] := Gm_sr[1];
      for i := 1 to 4 do Gstypes[1,i] := 0;
      GmTrust[1] := 0.0; Gm_se[1] := 0.0; Gm_sr[1] := 0.0;
      for pid := 1 to NAgents do begin
        i := 1;
        if apply_s_risk(pid) then i := i + 1;
        if apply_s_exploit(pid) then i := i + 2;
        Gstypes[1,i] := Gstypes[1,i] + 1;
        GmTrust[1] := GmTrust[1] + trust(pid);
        Gm_se[1] := Gm_se[1] + s_exploit(pid);
        Gm_sr[1] := Gm_sr[1] + s_risk(pid);
      end; { of pid Loop }
                                          GmSize[1]          :=
Int(countNC[1])/Int(NofRounds*NAgents);
      GCoop[1]  := Int(countNC[2])/Int(countNC[1])*100.0;
      GmTrust[1] := GmTrust[1]/Int(NAgents);
      Gm_se[1] := Gm_se[1]/Int(NAgents);
      Gm_sr[1] := Gm_sr[1]/Int(NAgents);
   end; { of CalcIndex }

{***** Data Writing Out *****}
 procedure DataOut;
   var
     StrategyRec  : AgentRec;
     SummaryRec   : SumRec;
     i, j, k      : integer;
   begin
      for i := 1 to NAgents do begin
        StrategyRec.profit := Agent[i].profit;
        for j := 1 to 19 do
          StrategyRec.strategy[j] := Agent[i].strategy[j];
        Write(Out_Agent, StrategyRec);
      end; { of i Loop }
      for i := 1 to 4 do
        SummaryRec.stypes[i] := Gstypes[1,i];
      SummaryRec.msize := GmSize[1];
      SummaryRec.Coop  := GCoop[1];
      SummaryRec.mtrust:= GmTrust[1];
      SummaryRec.m_se  := Gm_se[1];
      SummaryRec.m_sr  := Gm_sr[1];
      Write(Out_Summary, SummaryRec);
   end; { of DataOut }

{***** Drawing the Evolution Process per Round *****}
 procedure DrawProcess;
   const
      Ux    = 7;
      Uy    = 2;
      NofStr = 4;
   var
      i, j, k      : integer;
      ipage, igener : integer;
      CumCnt       : array[1..2] of integer;
      Gxy          : array[1..2,1..2] of integer;
      xy           : array[1..4] of TPoint;
   begin
      ipage  := (iGeneration -1) div 50 + 1;
      igener := iGeneration mod 50; if igener = 0 then igener
:= 50;
      if igener = 1 then begin
        with ProcessBox.Canvas do begin
          Pen.Color := clWhite; Brush.Color:= clWhite;
          Rectangle(0, 0, Width, Height);
          Pen.Color := clBlack;
          {Font.Name := 'Times New Roman';} Font.Style :=
[fsBold];
          Font.Color := clNavy; Font.Size := 11;
             TextOut(ProcessSize[1] div 2 -50, 2, 'Evolution
Process');
          Pen.Color := clBlack;
          MoveTo(10, 30); LineTo(10, ProcessSize[2]-20);
                              MoveTo(10,   ProcessSize[2]-20);
LineTo(ProcessSize[1]-10, ProcessSize[2]-20);
          Font.Color := clBlack; Font.Size := 10;
          TextOut(10,16, 'frequency');
                              TextOut(ProcessSize[1]  div  2-30,
ProcessSize[2]-15, 'Generation');
          {Pen.Color := clRed;
          MoveTo(Gf[3,1,1]-5, (Gf[3,1,2]+Gf[3,2,2]) div 2);
          LineTo(Gf[3,2,1], (Gf[3,1,2]+Gf[3,2,2]) div 2);}
        end; { of with ProcessBox.Canvas }
      end; { of if }
      for i := 1 to 2 do CumCnt[i] := 0;
      if iGeneration = 1 then
      begin
      for i := 1 to NofStr do
        if (Gstypes[1,i]>0) or (Gstypes[2,i]>0) then begin
          Gxy[1,1] := 10 + (igener-1)*Ux;
          Gxy[2,1] := 10 + igener*Ux;
          Gxy[1,2] := 30 + CumCnt[2]*Uy;
          Gxy[2,2] := 30 + CumCnt[1]*Uy;
          xy[1] := Point(Gxy[1,1], Gxy[1,2]);
          xy[2] := Point(Gxy[2,1], Gxy[2,2]);
               for j := 1 to 2 do CumCnt[j] := CumCnt[j] +
Gstypes[j,i];
          Gxy[1,2] := 30 + CumCnt[2]*Uy;
          Gxy[2,2] := 30 + CumCnt[1]*Uy;
          xy[4] := Point(Gxy[1,1], Gxy[1,2]);
          xy[3] := Point(Gxy[2,1], Gxy[2,2]);
          j := i;{ if i > NofSG then j := i - NofSG;}
          with ProcessBox.Canvas do begin
            Pen.Color := Scolor[j];
                                  MoveTo(Gxy[2,1],Gxy[2,2]);
LineTo(Gxy[2,1]+3,Gxy[2,2]);
          end; { of with ProcessBox.Canvas }
        end { of if }
      end
      else
      begin
      for i := 1 to NofStr do
        if (Gstypes[1,i]>0) or (Gstypes[2,i]>0) then begin
          Gxy[1,1] := 10 + (igener-1)*Ux;
          Gxy[2,1] := 10 + igener*Ux;
          Gxy[1,2] := 30 + CumCnt[2]*Uy;
          Gxy[2,2] := 30 + CumCnt[1]*Uy;
          xy[1] := Point(Gxy[1,1], Gxy[1,2]);
          xy[2] := Point(Gxy[2,1], Gxy[2,2]);
               for j := 1 to 2 do CumCnt[j] := CumCnt[j] +
Gstypes[j,i];
          Gxy[1,2] := 30 + CumCnt[2]*Uy;
          Gxy[2,2] := 30 + CumCnt[1]*Uy;
          xy[4] := Point(Gxy[1,1], Gxy[1,2]);
          xy[3] := Point(Gxy[2,1], Gxy[2,2]);
          j := i;{ if i > NofSG then j := i - NofSG;}
          with ProcessBox.Canvas do begin
            Pen.Color := Scolor[j]; Brush.Color := Scolor[j];
            Polygon(xy);
          end; { of with ProcessBox.Canvas }
        end; { of if }
      end;
   end; { of DrawProcess }

{***** Drawing Indices Change *****}
 procedure DrawIndices;
   const
      Ux    = 7;
```

```
    Uy    = 2;
    NofStr = 4;
  var
    i, j, k     : integer;
    ipage, igener : integer;
    CumCnt      : array[1..2] of integer;
    Gxy         : array[1..2,1..2] of integer;
    xy          : array[1..4] of TPoint;
  begin
    ipage  := (iGeneration -1) div 50 + 1;
     igener := iGeneration mod 50; if igener = 0 then igener
 := 50;
    if igener = 1 then begin
      with EventBox.Canvas do begin
        Pen.Color := clWhite; Brush.Color:= clWhite;
        Rectangle(0, 0, Width, Height);
        Pen.Color := clBlack;
        {Font.Name := 'Times New Roman';} Font.Style :=
[fsBold];
        Font.Color := clNavy; Font.Size := 11;
        TextOut(EventSize[1] div 2 -50, 2, 'Indices');
        Pen.Color := clBlack;
        MoveTo(10, 30); LineTo(10, EventSize[2]-20);
                              MoveTo(10,   EventSize[2]-20);
LineTo(EventSize[1]-10, EventSize[2]-20);
        Font.Color := clBlack; Font.Size := 10;
        TextOut(10,16, 'Value');
           TextOut(EventSize[1] div 2-30, EventSize[2]-15,
'Generation');
         {Pen.Color := clRed;
         MoveTo(Gf[3,1,1]-5, (Gf[3,1,2]+Gf[3,2,2]) div 2);
         LineTo(Gf[3,2,1], (Gf[3,1,2]+Gf[3,2,2]) div 2);}
      end; { of with EventBox.Canvas }
    end; { of if }
   {--GmSize--}
       Gxy[2,1] := 10 + (igener-1)*Ux; Gxy[1,1] := 10 +
(igener)*Ux;
    for i := 1 to 2 do
      Gxy[i,2] := 220 - Round(GmSize[i]*10.0);
    if iGeneration = 1 then
    with EventBox.Canvas do begin
            Pen.Color := clRed; Brush.Color := clRed;
Pen.Width := 1;
                              ellipse(Gxy[1,1]-3,Gxy[1,2]-3,
Gxy[1,1]+3,Gxy[1,2]+3);
    end { of with EventBox.Canvas }
    else
    with EventBox.Canvas do begin
      Pen.Color := clRed; Pen.Width := 3;
          MoveTo(Gxy[2,1], Gxy[2,2]); LineTo(Gxy[1,1],
Gxy[1,2]);
       Pen.Width := 1;
    end; { of with EventBox.Canvas }
      Edit5.Text := FloatToStrF(GmSize[1],ffFixed, 5,2);
Edit5.Refresh;
   {--GCoop--}
       Gxy[2,1] := 10 + (igener-1)*Ux; Gxy[1,1] := 10 +
(igener)*Ux;
    for i := 1 to 2 do
      Gxy[i,2] := 220 - Round(GCoop[i]*2.0);
    if iGeneration = 1 then
    with EventBox.Canvas do begin
            Pen.Color := clBlue; Brush.Color := clBlue;
Pen.Width := 1;
                              ellipse(Gxy[1,1]-3,Gxy[1,2]-3,
Gxy[1,1]+3,Gxy[1,2]+3);
    end { of with EventBox.Canvas }
    else
    with EventBox.Canvas do begin
      Pen.Color := clBlue; Pen.Width := 3;
          MoveTo(Gxy[2,1], Gxy[2,2]); LineTo(Gxy[1,1],
Gxy[1,2]);
       Pen.Width := 1;
    end; { of with EventBox.Canvas }
       Edit6.Text := FloatToStrF(GCoop[1],ffFixed, 6,2);
Edit6.Refresh;
   {--GmTrust--}
       Gxy[2,1] := 10 + (igener-1)*Ux; Gxy[1,1] := 10 +
(igener)*Ux;
    for i := 1 to 2 do
      Gxy[i,2] := 220 - Round((1.0-GmTrust[i])*200.0);
    if iGeneration = 1 then
    with EventBox.Canvas do begin
          Pen.Color := clGreen; Brush.Color := clGreen;
Pen.Width := 1;
                              ellipse(Gxy[1,1]-3,Gxy[1,2]-3,
Gxy[1,1]+3,Gxy[1,2]+3);
    end { of with EventBox.Canvas }
    else
    with EventBox.Canvas do begin
      Pen.Color := clGreen; Pen.Width := 3;
          MoveTo(Gxy[2,1], Gxy[2,2]); LineTo(Gxy[1,1],
Gxy[1,2]);
       Pen.Width := 1;
    end; { of with EventBox.Canvas }
       Edit7.Text := FloatToStrF(1.0-GmTrust[1],ffFixed,
5,2); Edit7.Refresh;
   {--Gm_se--}
       Gxy[2,1] := 10 + (igener-1)*Ux; Gxy[1,1] := 10 +
(igener)*Ux;
    for i := 1 to 2 do
      Gxy[i,2] := 220 - Round(Gm_se[i]*200.0);
    if iGeneration = 1 then
    with EventBox.Canvas do begin
        Pen.Color := clFuchsia; Brush.Color := clFuchsia;
Pen.Width := 1;
                              ellipse(Gxy[1,1]-3,Gxy[1,2]-3,
Gxy[1,1]+3,Gxy[1,2]+3);
    end { of with EventBox.Canvas }
    else
    with EventBox.Canvas do begin
      Pen.Color := clFuchsia; Pen.Width := 1;
          MoveTo(Gxy[2,1], Gxy[2,2]); LineTo(Gxy[1,1],
Gxy[1,2]);
       Pen.Width := 1;
    end; { of with EventBox.Canvas }
       Edit8.Text := FloatToStrF(Gm_se[1],ffFixed, 5,2);
Edit8.Refresh;
   {--Gm_sr--}
       Gxy[2,1] := 10 + (igener-1)*Ux; Gxy[1,1] := 10 +
(igener)*Ux;
    for i := 1 to 2 do
      Gxy[i,2] := 220 - Round(Gm_sr[i]*200.0);
    if iGeneration = 1 then
    with EventBox.Canvas do begin
            Pen.Color := clNavy; Brush.Color := clNavy;
Pen.Width := 1;
                              ellipse(Gxy[1,1]-3,Gxy[1,2]-3,
Gxy[1,1]+3,Gxy[1,2]+3);
    end { of with EventBox.Canvas }
    else
    with EventBox.Canvas do begin
      Pen.Color := clNavy; Pen.Width := 1;
          MoveTo(Gxy[2,1], Gxy[2,2]); LineTo(Gxy[1,1],
Gxy[1,2]);
       Pen.Width := 1;
    end; { of with EventBox.Canvas }
       Edit9.Text := FloatToStrF(Gm_sr[1],ffFixed, 5,2);
Edit9.Refresh;
  end; { of DrawIndices }

{*****  Draw Graphs  *****}
 procedure Graphs;
  var
    i, j, k  : integer;
    pid      : byte;
  begin
    DrawProcess;
    DrawIndices;
  end; { of Graphs }

{*****  Strategy Change  *****}
 procedure StrategyChange;
  const
```

```
      Nchange        = 10;  { N of strategy to be changed,
upper & lower }
  var
    Uid, Lid      : array[1..2, 1..Nchange] of integer;

  procedure Selection;
   var
     i, j, k, L, m   : integer;
     p_id            : b_vec;
     payment         : r_vec;
     mean, ss        : reaL;
     max, min, dum   : reaL;
     imax, imin, idum : integer;
     done            : boolean;
     tieid           : b_vec;
     ntie            : integer;
     Sid, Eid        : integer;
     scnt            : integer;
     tempvec1        : dig2;
     tempvec2        : dig2;
     ClassSize, classid : integer;
     str50           :string[50];
   begin
      for i := 1 to NAgents do
      if Agent[i].profit < 0.0 then begin
                                  str50  :=  IntToStr(i)  +  ':'  +
FloatToStrF(Agent[i].profit,ffFixed, 20,3);
        form1.memo1.lines.add(str50);
      end; { of i Loop }
    ClassSize := NAgents{ div 2};
    for classid := 1 to 1{2} do begin
      Sid := (Classid-1)*ClassSize + 1;
      Eid := Classid*ClassSize;
    {+++  variance judgment  +++}
      mean := 0.0; ss := 0.0;
      for i := Sid to Eid do begin
        mean := mean + Agent[i].profit;
        ss  := ss  + Agent[i].profit*Agent[i].profit;
      end; { of i Loop }
      mean := mean/Int(ClassSize);
      ss := ss - Int(ClassSize)*mean*mean;
      ss := ss / Int(ClassSize-1);
    if ss > 0.0 then begin
    {+++  Re-Ordering  +++}
      for i := 1 to NAgents do begin
        p_id[i] := i;
        payment[i] := Agent[i].profit;
      end; { of i loop }
      for i := Sid to Eid-1 do begin
        max := payment[i]; imax := p_id[i];
        for j := i to Eid do
          if payment[j] > max then begin
              dum := max; max := payment[j]; payment[j] :=
dum;
            payment[i] := max;
            idum := imax; imax := p_id[j]; p_id[j] := idum;
            p_id[i] := imax;
          end; { of if }
      end; { of i loop }
    {+++  seek upper  +++}
      max := payment[Sid+Nchange-1];
      k := Sid + Nchange - 1; done := false;
      repeat
          k := k - 1;
          if k = Sid - 1 then done := true;
          if k > Sid - 1 then
            if payment[k] > max then done := true;
      until done;
      k := k + 1;
      L := Sid + Nchange - 1; done := false;
      repeat
          L := L + 1;
          if payment[L] < max then done := true;
          if L = Eid then done := true;
      until done;
      L := L - 1; if L = Eid then L := L + 1;
      ntie := L - k + 1;
      if k > Sid then
            for i := Sid to k-1 do Uid[ClassId, i-Sid+1] :=
p_id[i];
      for i := k to L do tieid[i-k+1] := p_id[i];
      L := 0;
      repeat
          i := random(ntie) + 1;
          if tieid[i] <> 0 then begin
            L := L + 1;
            Uid[ClassId, k-Sid+L] := tieid[i];
            tieid[i] := 0;
          end; { of if }
      until k-Sid+L = Nchange;
    {+++  seek lower  +++}
      min := payment[Eid-Nchange+1];
      k := Eid - Nchange + 1; done := false;
      repeat
          k := k + 1;
          if k = Eid+1 then done := true;
          if k < Eid+1 then
            if payment[k] < min then done := true;
      until done;
      k := k - 1;
      L := Eid - Nchange + 1; done := false;
      repeat
          L := L - 1;
          if L = Sid - 1 then done := true; {DDDDD}
          if payment[L] > min then done := true;
      until done;
      L := L + 1;
      ntie := k - L + 1;
      if k < Eid then
         for i := Eid downto k+1 do Lid[ClassId, Eid-i+1] :=
p_id[i];
      for i := k downto L do tieid[k-i+1] := p_id[i];
      L := 0;
      repeat
          i := random(ntie) + 1;
          if tieid[i] <> 0 then begin
            L := L + 1;
            Lid[ClassId, Eid-k+L] := tieid[i];
            tieid[i] := 0;
          end; { of if }
      until Eid-k+L = Nchange;
    {+++  shuffle  +++}
      {for i := 1 to Nchange do begin
        k := Uid[ClassId, i]; L := Lid[ClassId, i];
        for j := 1 to 19 do
          Agent[L].strategy[j] := Agent[k].strategy[j];
      end;} { of i loop }
    end; { of if ss > 0.0 }
    end; { of clanid loop }
  end; { of Selection }

  procedure CrossOver;
   var
     i, j, k, L, m    : integer;
     ico              : integer;
     payment          : r_vec;
     TotalProfit      : reaL;
     dice, jvalue     : reaL;
     jid              : array[1..2] of byte;
     done             : boolean;
     cutline          : byte;

   procedure LowerShuffle;
    var
      i, j, k    : integer;
      seL1, seL2 : integer;
    begin
       for i := 1 to 10 do
         for j := 1 to NChange do begin
                    seL1  :=  random(NChange)+1;  seL2  :=
random(NChange)+1;
           k := Lid[1,seL1];
           Lid[1,seL1] := Lid[1,seL2];
           Lid[1,seL2] := k;
```

```
            end; { of j Loop }
        end; { of LowerShuffle }

      begin
          LowerShuffle;
          for i := 1 to NChange do
            payment[i] := Agent[Uid[1,i]].profit;
          if payment[NChange] < 0.0 then
            for i := 1 to NChange do
                                    payment[i] := payment[i] +
1.5*Abs(payment[NChange]);
          TotalProfit := 0.0;
          for i := 1 to NChange do
            TotalProfit := TotalProfit + payment[i];
          for ico := 1 to NChange div 2 do begin
            for i := 1 to 2 do begin
              dice := random;
              jvalue := 0.0; jid[i] := 0; done := false;
              repeat
                  jid[i] := jid[i] + 1;
                  jvalue := jvalue + payment[jid[i]]/TotalProfit;
                  if jvalue >= dice then done := true;
              until done or (jid[i] = NChange);
            end; { of i Loop }
            cutline := random(18) + 1;
            for i := 1 to cutline do
              Agent[Lid[1,(ico-1)*2+1]].strategy[i]
                := Agent[Uid[1,jid[1]]].strategy[i];
            for i := cutline+1 to 19 do
              Agent[Lid[1,(ico-1)*2+1]].strategy[i]
                := Agent[Uid[1,jid[2]]].strategy[i];
            for i := 1 to cutline do
              Agent[Lid[1,ico*2]].strategy[i]
                := Agent[Uid[1,jid[2]]].strategy[i];
            for i := cutline+1 to 19 do
              Agent[Lid[1,ico*2]].strategy[i]
                := Agent[Uid[1,jid[1]]].strategy[i];
          end; { of ico Loop }
      end; { of CrossOver }

    procedure Mutation;
      var
        i, j, k     : integer;
        NewStrategy  : integer;
        scnt        : integer;
      begin
          if DoMutant then
          if not converge then
          for i := 1 to NAgents do begin
            for j := 1 to 19 do
              if random < Pmutant then
                                  if Agent[i].strategy[j] = 0 then
Agent[i].strategy[j] := 1
                                    else Agent[i].strategy[j] := 0;
          end; { of i Loop }
                {for j :=  5 to  5 do Agent[i].strategy[j] := 1;}
{NNNNN}
                {for j :=  8 to  8 do Agent[i].strategy[j] := 1;}
{NNNNN}
          if not bribe then
                for j := 16 to 17 do Agent[i].strategy[j] := 0;
{NNNNN}
      end; { of Mutation }

    begin
        Selection;
        CrossOver;
        Mutation;
    end; { of StrategyChange }

{*****  Results  *****}
  procedure Results;

    procedure CloseFiles;
      begin
          CloseFile(Out_Agent); CloseFile(Out_Summary);
      end; { of CloseFiles }

    begin
        CloseFiles;
    end; { of Results }

{*****  Finale  *****}
  procedure Finale;
    begin
        form1.memo1.lines.add('Normal End of Job');
    end; { of Finale }

{=====  Main Routine  =====}
begin
    Global_Init;
    for iCond := 0 to 3 do
    for iSim := SSim to ESim do begin
      Sim_Init;
     {Generation Starts}
      repeat
          Generation_Init;
         {Round Starts}
          repeat
              Round_Init;
              Gathering_Phase;
              {Account;}
          until iRound >= NofRounds;
         {Round Ends}
          CalcIndex;
          DataOut;
          Graphs;
            if iGeneration = NofGenerations then converge :=
true;
          if not converge then StrategyChange;
      until converge;
     {Generation Ends}
      Results;
    end; { of iSim Loop }
    Finale;
end;

end.
```