Ascape を利用して

—— エージェントベースモデルの開発・探索ツール ——

社会情報学科　行　方　常　幸

目　次

1．はじめに

われわれの社会は行動主体である個人が多数集まり，各々が他と関わり行為を行い，その総体として社会現象が生じている，と見なすことができる。エージェントベースモデルはこの考えを単純化した状況に適用し，個人（エージェント）の行為と社会現象の関係を調べる研究方法である。本稿では，2次元格子上に配置されたエージェントが囚人のジレンマゲームを行う状況を例にとり，エージェントベースモデルの開発・探索ツールである Ascape の利用法を

概観する。

2．Ascape とは？

Ascape（http://ascape.sourceforge.net/）はエージェントベースモデルを容易に開発・探索するツールである。Java 言語で書かれており，無料で利用可能である。多くの例が附属しており，それを参照することにより，Java 言語を利用してプログラムをした経験のある人なら，Ascape の細かい部分に関する知識があまりなくても，概ね，開発したいエージェントベースモデルに関する部分のみをプログラムすることにより，比較的簡単にシミュレーションを行うことができる。また，コード部分を変更することなく，シミュレーションを実行中にパラメータの値を変えることができ，パラメータの変化によりシミュレーションの結果がいかに変化するかを容易に探索可能である。

Ascape は「エイスケイプ」と発音され，A は Agent を Scape は landscape の scape を指し，エージェントにより何かが起きるある種の空間，を意味するとのことである。Ascape では IDE（Integrated Development Environment）として Eclipse（http://eclipse.org/）を推奨しているが，本稿では NetBeans（http://netbeans.org/）を利用する。

まず，Ascape を「http://sourceforge.net/projects/ascape/files/」からダウンロードする。ページの下の方，All Files の「Ascape」の（執筆時点での）最新バージョン「5.6.0」を開き

Ascape_5.6.0.jar

Ascape_Docs_5.6.0.zip

Ascape_SDK_5.6.0.zip

Ascape_Applet5.6.0.jar（アプレットで実行する場合）

をダウンロードし，自分の好きなディレクトリに保存しておく。

3．モ デ ル

本稿で扱うモデルに関して簡単に説明する（詳しくは[3]を参照）。第0期にN（＝100）人のプレイヤー（今後はエージェントではなくプレイヤーという用語を利用する）が30×30のセルからなる2次元格子上にランダムに配置される。この2次元格子の上の境界と下の境界は，また，左の境界と右の境界は，繋がっている。プレイヤーは3つのタイプ，Referential（Ref.と略記する），Cooperator（Coop.），Defector（Def.）からなる。プレイヤーは表1で与えられる2人囚人のジレンマゲームを行う。Def.は何時もDを取り，Coop.は何時もCを取る。プレイヤー，特に，Ref.は相手をタグにより区別する。タグの個数はnumberOfTags（＝4）個で，0,…,numberOfTags−1の整数値を取る。Ref.の戦略は少々複雑である。Ref.はタグに基づくTFTを利用する。すなわち，あるタグを持つ相手に最初に会った時はCを取り，同じタグに2回目以降に会った時はそのタグを持つ直前の相手が取った手を取る。更に，Ref.は相手が人脈に入っていれば，Cを取る。人脈に関しては次の段落で述べる。プレイヤーが最初に保持している富はinitialWealth（＝6）である。第一世代のプレイヤーの年齢は0以上deathAge（＝50）以下のランダムな整数値である。

表1

	C	D
C	R,R	S,T
D	T,S	P,P

$T=6, R=5, P=-5, S=-6.$

各期において，各プレイヤーは（1）移動し，そして（2）他のプレイヤーと表1で与えられる2人囚人のジレンマゲームを行う（表2を参照）。共にCを取った場合，相手を自分の人脈に入れる。人脈には最近のlengthOfConnections（＝10）人のみが保持される。ゲームの結果により得られた利得は富に加えられる。富がfissionWealth（＝10）を超え，フォン・ノイマン近傍に空いているセルがあれば子供を産み，その子供に自分の富の中からinheritedWealth（＝6）を与える。富が負になり（表2の（3）で述べられている）人脈の助けでも非負の富にならない場合は，死に，2次元格子から取り除かれ

る。富が非負の場合，年齢を1だけ増やす。年齢がdeathAge（＝50）を超えても，死ぬ（表2の（3）を参照）。その後，次の期が始まる。

表2

(1)	確率rateOfGlobalMoveToLocalでプレイヤーは2次元格子全体の空いているランダムなセルに移動する。もしこのようなセルがなければ現在のセルに留まる。または，確率1-rateOfGlobalMoveToLocalでフォン・ノイマン近傍をランダムに選び，そのセルが空いていれば移動する。もしそのようなセルがなければ現在のセルに留まる。
(2)	確率rateOfGlobalInteractionToLocalでゲームを行う相手が（自分を除く）すべてのプレイヤーからランダムに選ばれる。確率1-rateOfGlobalInteractionToLocalでゲームを行う相手がフォン・ノイマン近傍にいるプレイヤーからランダムに選ばれる。この近傍に誰もいなければゲームを行わない。
(3)	Ref.とCoop.のみが人脈を持つ。人脈の人数が正の場合，(TakeWealthFromConnections, GiveWealthWhenDie) = (false, false)，(true, false)，または（true, true）の3通りに人脈を利用する。 TakeWealthFromConnections＝trueなら：ゲームの後に富が負になれば，自分の人脈内の生きているプレイヤーに1個ずつ富をもらうことにより，自分の富が0以上になるか否かをチェックする。もし0以上になれば実際に貰い，この期に死なない。もし，0未満ならばこの期に死ぬ。 TakeWealthFromConnections＝trueに加えGiveWealthWhenDie＝trueなら：寿命が尽きて死ぬ時，自分の富が正ならば（その中から最大で20個まで）自分の人脈内の生きているプレイヤーになるべく等しく分け与える。

親から子供へ，親の性質であるタイプ，タグ，rateOfGlobalMoveToLocal，rateOfGlobalInteractionToLocalが遺伝する。ただし，小さな確率mutationRate（＝0.05）で突然変異が起こる。突然変異が起こる場合と，第一世代のこれらの性質の初期分布は次の表3で与えられる。

以上が，本稿で扱うモデルの概要である。以降，このモデルをPDGameと呼ぶことにする。このPDGameをAscapeを用いて実装する。

表 3

性　質	初　期　分　布
タイプ	第一世代は確率 referentialRatio で Ref.，確率 cooperatorRatio で Coop.，残りの確率で Def. になる。 突然変異の時は，第一世代に存在していたタイプに等確率でなる。
タグ	等確率で {0,..., numberOfTags-1} のどれかになる。
rateOfGlobalMoveToLocal	次の区間内での一様分布：[lowRateOfGlobalMoveToLocal, highRateOfGlobalMoveToLocal].
rateOfGlobalInteractionToLocal	次の区間内での一様分布：[lowRateOfGlobalInteractionToLocal, highRateOfGlobalInteractionToLocal].

4．Ascape による実装

まず，「2．Ascape とは？」でダウンロードした Ascape 関連のファイルを NetBeans に認識させる。NetBeans を起動して，メニューから［ツール］-［ライブラリ］を選び適当な名前（MyLibrary）で新規のライブラリを登録する。MyLibrary を選択しクラスパスに Ascape_5.6.0.jar を，ソースに（Ascape_SDK_5.6.0.zip からソースファイルだけを抽出して作成した（詳しくは，補遺を参照））Rev_Ascape_Source_5.6.0ディレクトリを，Javadoc に Ascape_Docs_5.6.0.zip を設定する。これで MyLibrary に Ascape の必要なファイルが設定された。特に，ソースと Javadoc を設定したことにより，NetBeansIDE の中から Ascape のソースコードと Javadoc（API の定義など）が参照可能となり，プログラミングが容易になる。

次に，Java アプリケーション用のプロジェクト（MyAscapeModels としておく）を作成する。先程登録した MyLibrary をこのプロジェクトのライブラリに設定する。具体的には，プロジェクトウィンドウでプロジェクト名 MyAscapeModels を右クリックし，プロパティを選び，表示されるダイアログボックスの左にあるカテゴリからライブラリを選び，右にある「ライブラリを追加」ボタンを押し，MyLibrary を追加する。これで Ascape を利用する

準備が整った。以下で，このMyAscapeModelsプロジェクトの中に「3．モデル」のところで述べたモデルPDGameを実装する。以下で作成するPDGameを実行すると図1のようになる。ウィンドウの上部の左から「Viewの選択」，「停止中」，「第43期」，等が示され，その右にコントロールボタンが並んでいる。その下の左上から右下へ，2次元格子，データを表わすグラフ，が並んでいる。

Ascapeでエージェントベースモデルを実装するには，最低限2つのファイル（クラス）が必要である。モデルを記述するファイル（クラス）とプレイヤーを記述するファイル（クラス）である。PDGameモデルを記述するクラスを

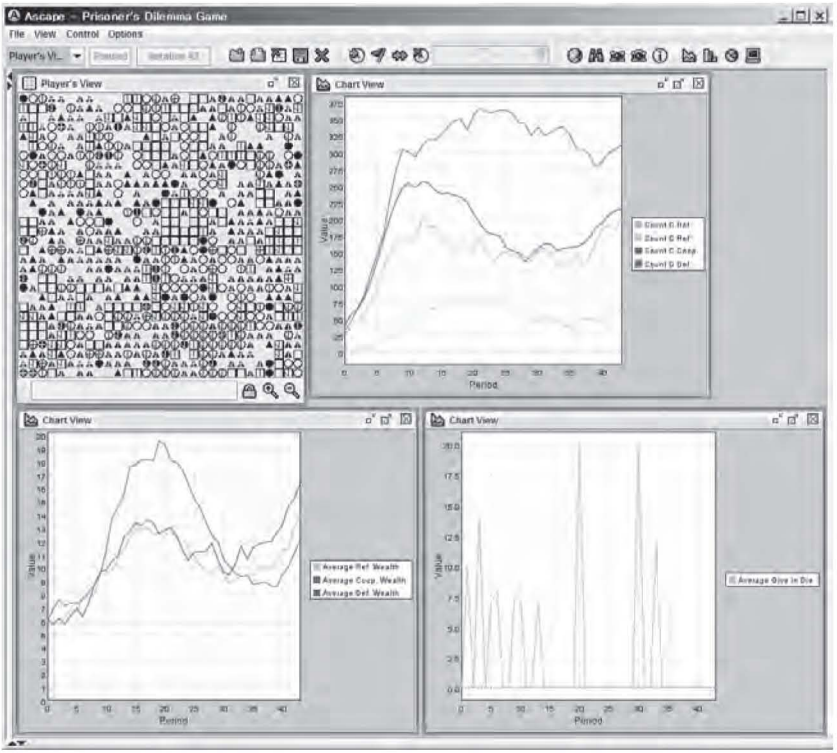

図1　PDGameの実行ウィンドウ

PDGame（ファイル名はPDGame.java）とし，そのプレイヤーを記述するクラスをPDGamePlayer（ファイル名はPDGamePlayer.java）とすることにする。PDGameクラスはScapeクラスの派生クラス，PDGamePlayerクラスはCellOccupantクラスの派生クラスとする。以下では，最初にPDGameクラス，次に，PDGamePlayerクラスに関して述べる。

1 PDGameクラス（モデルを記述するクラス）

プロジェクトウィンドウでプロジェクト名MyAscapeModelsを右クリックし，［新規］-［Javaクラス...］を選ぶ。表示されるダイアログボックスでクラス名を「PDGame」，パッケージを（ここでは）「org.namekata.ascape.models.pd」とし，完了ボタンを押す。ファイルPDGame.javaが作成され編集可能になる。クラス名PDGameの右に「extends Scape」を加えPDGameクラスをScapeクラスの派生クラスにすることを明記する。（適宜，右クリックして「インポートを修正」を選び必要なimport文を入れる必要があるが，これに関しては，今後省略する。）後は，修正したいsuperクラスScapeのメソッドをPDGameクラス内に記述し，シミュレーション実行中にパラメータの変更を可能にするために，getメソッドとsetメソッドを記述するだけである。以下に出てくるコードの抜粋部分で未宣言の変数はクラス内の適切な場所で適切に宣言されている。

getメソッドとsetメソッド以外に記述する必要があるメソッドは，createScape()，scapeSetup(ScapeEvent)，createGraphicViews()である。本稿では，更に，main(String[])を記述した。

モデルを記述するPDGameクラスで行うべきことは，以下の通りである。

createScape()メソッドにて：（コード1を参照）

まず，superメソッドを呼ぶ。

次に，（本稿では）latticeというScape変数にフォン・ノイマン近傍を持ち，各セル（PrototypeAgent）にセルの色を緑色から変更するためだけに（普通の）HostCellクラスを派生させた（後述する）MyHostCellを指定した，

```
//コード1（PDGame.java）

// createScape()と scapeSetup(ScapeEvent)

public void createScape() {
    super.createScape();
    setName("Prisoner's Dilemma Game");
    lattice = new Scape();
    lattice.setSpace(new Array2DVonNeumann());
    lattice.setPrototypeAgent(new MyHostCell());
    lattice.setName("Lattice");
    lattice.setExtent(latticeWidth, latticeHeight);

    PDGamePlayer player = new PDGamePlayer();
    player.setHostScape(lattice);
    players = new Scape();
    players.setPrototypeAgent(player);
    players.setExecutionOrder(Scape.RULE_ORDER);
    players.setName("Players");

    add(lattice);
    add(players);

    //（C1）シミュレーション実行中にデータを集めたい場合、
    //ここに、後述する StatCollector を定義し、
    //それを players に加える。
}

public void scapeSetup(ScapeEvent scapeEvent) {
    players.setExtent(nPlayers);
}
```

2次元格子を設定し，格子のサイズを指定する。

その次に，後で作成するPDGamePlayerクラスのインスタンスを（本稿では）playerという変数に作成し，それが活動するHostScapeを今作成した2次元格子latticeに指定する。lattice上で活動するすべてのプレイヤーを保持するScape変数playersを作成し，そのPrototypeAgentをplayerに指定する。Scape変数playersに設定されるルールの実行順序（これに関しては後述する）をRULE_ORDER（または，AGENT_ORDER）に指定する。この実行順序はシミュレーション実行時にSetting Windowで変更可能である。playersに保持する第一世代のプレイヤーの総数は次に述べるscapeSetup(ScapeEvent) メソッドで記述する。

上記に設定した2次元格子を表わすlatticeとプレイヤー全体を表わすplayersの2つのScapeを，これもScapeである自分自身（PDGame）に加

える（add）。

最後の（C1）の部分は，シミュレーション実行中にデータを集めたい場合に必要な部分で，これに関しては後述する。

scapeSetup(ScapeEvent) メソッドにて：（コード 1 を参照）

players に保持する第一世代のプレイヤーの総数 nPlayers を players Scape に設定する。

createGraphicViews() メソッドにて：（コード 2 を参照）

シミュレーションを視覚的に見ることができるようにする部分である。

まず，super メソッドを呼ぶ。

2 次元格子上においてプレイヤーがどのように移動しどのような行為を取っているか？を視覚的に見るために，それに適した Overhead2DView を作成し，セルのサイズを設定し，2 次元格子を表わす Scape の lattice に加える（addView）。

（C2）の部分はシミュレーション中に集めたデータ（コード 1 の（C1）の部分で定義したもの）を視覚的に表示する時に必要な部分であり，後述する。

（C3）の部分は 2 次元格子上で表示されるプレイヤーの形を自分の好きなように変える時に必要な部分であり，後述する。

```
//コード 2（PDGame.java）
public void createGraphicViews() {
    super.createGraphicViews();

    overheadView = new Overhead2DView("Player's View");
    overheadView.setCellSize(12);
    lattice.addView(overheadView);

    //（C2）シミュレーション実行中に StatCollector で集めたデータの表示

    //（C3）プレイヤーの描画方法を自分で変える場合
}
```

set メソッドと get メソッド：（コード 3 を参照）

シミュレーションの実行中に値を変更したいパラメータには set メソッドと get メソッドを書いておく必要がある。例えば，第一世代のプレイヤーの

人数はnPlayerという変数で与えられるが，シミュレーション中に変更できるようにするにはコード3の該当部分のようにする。こうすることでSetting Windowで変更可能となる。

main(String[])メソッドにて：（コード3を参照）

プロジェクトMyAscapeModelsを（例えば，MyAscapeModels.jarファイルをダブルクリックする，または，アプレットのarchiveでこのjarファイルを指定して）起動した時にPDGameモデルが実行されるようにするには，まず，実行時の主クラスをPDGameに指定する。具体的には，プロジェクトウィンドウでプロジェクト名MyAscapeModelsを右クリックし「プロパティ」を選び，左のカテゴリから「実行」を選び，右の「主クラス」の参照ボタンを押し，「org.namekata.ascape.models.pd.PDGame」を設定する。次に，mainメソッド内でコード3の該当部分のようにする。

```
//コード 3（PDGame.java）
public void setNPlayers(int nPlayers) {
    this.nPlayers = nPlayers;
}

public int getNPlayers() {
    return nPlayers;
}

public static void main(String[] args) {
    new org.ascape.runtime.swing.SwingRunner()
            .open("org.namekata.ascape.models.pd.PDGame", new String[]{});
}
```

mainメソッドを記述しない場合は，「主クラス」に「org.ascape.runtime.swing.SwingRunner」を指定する。jarファイルを実行すると，実行するモデルのクラス名を入力するか？モデルをリストから選ぶか？のどちらかを行わなければならない。前者のクラス名を入力する場合，正確に「org.namekata.ascape.models.pd.PDGame」と入力しなければならない。また，モデルを選ぶ候補のリストはjarファイル中の「org/ascape/model/ModelChoices.txt」に記述されており，Ascapeに添付されているモデルが列挙されてい

る。PDGameをリストに表示させるためには，このModelChoices.txtを修正すればよい（補遺参照）。

以上が，PDGameクラス（モデルを記述するクラス）の説明である。

2 PDGamePlayerクラス（プレイヤーを記述するクラス）

プロジェクトウィンドウでプロジェクト名MyAscapeModelsを右クリックし，［新規］－［Javaクラス...］を選ぶ。表示されるダイアログボックスでクラス名を「PDGamePlayer」，パッケージを（ここでは）「org.namekata.ascape.models.pd」とし，完了ボタンを押す。ファイルPDGamePlayer.javaが作成され編集可能になる。クラス名PDGamePlayerの右に「extends CellOccupant」を加えPDGamePlayerクラスをCellOccupantクラスの派生クラスにすることを明記する。後は，修正したいスーパークラスCellOccupantのメソッドをPDGamePlayerクラス内に記述し，その他必要なことをすればよい。本稿で記述したメソッドは，clone()，initialize()，scapeCreated()，play(Agent)，fissionCondition()，fission()，update()，deathCondition()，die()である。また，プレイヤーの移動ルールを定義したRuleクラスの定数GLOCAL_MOVE_RULE，プレイヤーの対戦相手を決めるルールを定義したRuleクラスの定数GLOCAL_INTERACTION_RULE，等を記述した。

clone()メソッドは第一世代のプレイヤーや子供を産む時に呼び出されるメソッドであり，PDGamePlayerクラス内で利用しているクラス変数があれば，このclone()で新しいインスタンスを作成する。そうしないと複数の子孫が同じインスタンスを共有してしまう。一方，initialize()メソッドは第一世代のプレイヤーにのみに，clone()メソッドの次に呼び出されるメソッドである。従って，第一世代の性質の設定，例えば，タイプをRef.またはCoop.またはDef.のどれかに指定された確率で決める，タグを等確率で｛0,...，numberOfTags-1｝のどれかに決める，等を行う。

scapeCreated()メソッドではプレイヤーを保持するScape（今の場合はPDGameクラスの）playersにルールを設定する（コード4を参照）。

```
//コード 4（PDGamePlayer.java）
public void scapeCreated() {
    getScape().addInitialRule(MOVE_RANDOM_LOCATION_RULE);
    getScape().addInitialRule(INITIAL_CONNECTIONS);

    getScape().addRule(GLOCAL_MOVE_RULE);
    getScape().addRule(GLOCAL_MOVE_AVAILABLE_RULE, false);
    getScape().addRule(GLOCAL_INTERACTION_RULE);

    getScape().addRule(FISSIONING_RULE);
    getScape().addRule(UPDATE_RULE);
    getScape().addRule(DEATH_RULE);
}
```

GLOCAL_MOVE_RULE から DEATH_RULE までの 6 つのルールは，シミュレーション実行時に Setting Window で実行順序を変えたり，ルールの実行を無効にできる。GLOCAL_MOVE_AVAILABLE_RULE はこの時点で無効に設定されている。

ここで，コード 1 に出てきたルールの実行順序 RULE_ORDER と AGENT_ORDER の違いを述べる。RULE_ORDER の場合，GLOCAL_MOVE_RULE がすべてのプレイヤーに適用された後に，次の GLOCAL_INTERACTION_RULE がすべてのプレイヤーに適用され，その後に，FISSIONING_RULE がすべてのプレイヤーに適用され，…というように，各ルールがすべてのプレイヤーに適用されてから次のルールへ移る，という具合にルールが実行される。いわゆる，プレイヤーに対して同期的にルールが適用される。他方，AGENT_ORDER の場合，あるプレイヤーに対して，まず，GLOCAL_MOVE_RULE が適用され，その次に GLOCAL_INTERACTION_RULE が適用され…DEATH_RULE が適用される。次のプレイヤーに対して，まず，GLOCAL_MOVE_RULE が適用され，その次に GLOCAL_INTERACTION_RULE が適用され…DEATH_RULE が適用される，という具合に，各プレイヤー毎にすべてのルールが実行される。いわゆる，プレイヤーに対して非同期的にルールが適用される。

addInitialRule は最初に 1 度だけ実行するルールを指定する。MOVE_RANDOM_LOCATION_RULE は Ascape で定義されているルールでプレイ

ヤーを2次元格子上にランダムに配置する。(INITIAL_CONNECTIONSは自作のルールであるが，細かいことなので説明を省略する。) addRuleは毎期に実行するルールを指定する。GLOCAL_MOVE_RULEは表2の(1)を実装したルールであり，Ascapeで定義されているMOVE_RANDOM_LOCATION_RULEとRANDOM_WALK_RULEを参考に作成した。GLOCAL_MOVE_AVAILABLE_RULEも同様なルールであり，Ascapeで定義されているRANDOM_WALK_AVAILABLE_RULEを参考に作成した。GLOCAL_INTERACTION_RULEは表2の(2)を実装したルールであり，Ascapeで定義されているPLAY_OTHERとPLAY_RANDOM_NEIGHBOR_RULEを参考にして作成した。GLOCAL_INTERACTION_RULEが実行されると，対戦相手として選ばれたプレイヤーを引数としてplay(Agent)メソッドが呼ばれる。従って，このplay(Agent)メソッドで，表1の囚人のジレンマゲームによる自分と対戦相手の利得を計算し，各々の富に加え，その他必要な処理をする。FISSIONING_RULE，UPDATE_RULE，DEATH_RULEはAscapeで定義されているルールである。FISSIONING_RULEは子供を産むことに対応するルールで，このルールが実行されると，fissionCondition()が呼び出され，これが真を返すとfission()メソッドが呼ばれる。fissionCondition()メソッドでは富がfissionWealthよりも大きい場合は真を，そうでない場合は偽を返す。fission()メソッドでは子供を産むための場所が空いているかどうか調べ，空いている場合には，

```
PDGamePlayer child =(PDGamePlayer) this.clone() ;
getScape().add(child) ;
child.moveTo(cell) ; //cell は空いているセルである
```

のように子供を産み，この子供を，プレイヤーを保持するScape，(今の場合はPDGameクラスの) playersに加え，空いているセルに配置する。突然変異が起こらなければ，この子供に親から性質を遺伝させ，突然変異が起これば表3のようにランダムに設定する。UPDATEは年を一つ取るルールに対応し，

update() メソッドが呼ばれる。update() では年齢を 1 だけ増やす。DEATH_RULE は死ぬことに対応するルールで，このルールが実行されるとdeathCondition() がチェックされ，真を返すとプレイヤーは死ぬ。偽を返すと死なない。deathCondition() メソッドでは年齢が寿命を超えているか？または，富が負であり，かつ，人脈から富を貰っても富が負のままであるか？を調べ（表 2 の（3）参照），少なくとも一方が真である場合，真を返す。それ以外は偽を返す。

3　シミュレーション実行中にデータを集め表示する

シミュレーションを行う目的の一つは，ある事象が時間と共にどのように変化するか？を見ることである。例えば，シミュレーションの実行中に，C を取っている Ref. の人数，Ref. の富の平均，（表 2 の（3）で説明した）寿命で死ぬ時に人脈に与えた富の平均，を収集して折れ線グラフで表示する，を例に取り，その実装方法を述べる（コード（C1）と（C2）を参照）。コード（C1）と（C2）はモデルを記述する PDGame.java のコード 1 の該当部分に入る。コード（C1）はデータを収集する部分，コード（C2）はグラフで表示する部分である。

まず，「C を取っている Ref. の人数」を集計するために，StatCollectorCond というクラスを利用する。ある条件を満たした時のプレイヤーの個数（人数）を集計するクラスである。引数として自分で適切と思える名前（ここでは）「C Ref.」を与えて，無名の内部クラスを利用する。その内部クラスの meetsCondition の中でカウントしたい場合に真を返す return 文を書く。object にプレイヤーが入っているので PDGamePlayer にキャストして type が REFERENTIAL であり move が MOVE_C であるときに真を返す。

「Ref. の富の平均」を集計する場合は StatCollectorCondCSA というクラスを利用する。条件を満たした時のプレイヤーの個数（人数）を数え（Count），（getValue で返す）データの和（Sum），その値の平均（Average）を計算するクラスである。基本的なことであるが，シミュレーション中のある期に Ref. が Count 人いて，それらの富の和が Sum であるとき，富の平均 Average

```
                    //コード（C1）（PDGame.java）
StatCollector[] stats = {
    new StatCollectorCond("C Ref.") {

        public boolean meetsCondition(Object object) {
            return ((PDGamePlayer) object).type == REFERENTIAL
                && ((PDGamePlayer) object).move == MOVE_C;
        }
    },
// ...
    new StatCollectorCondCSA("Ref. Wealth") {

        public boolean meetsCondition(Object object) {
            return ((PDGamePlayer) object).type == REFERENTIAL;
        }

        public double getValue(Object object) {
            return ((PDGamePlayer) object).getWealth();
        }
    },
// ...
};

players.addStatCollectors(stats);
players.addStatCollector(new StatCollectorCSA("Give in Die", false));
```

は Average＝Sum/Count である。適切な名前（ここでは）「Ref. Wealth」を付け，meetsCondition で type が REFERENTIAL であるときに真を返し，getValue でデータの値（今の場合は，プレイヤーの富）((PDGamePlayer) object).getWealth() を返す。以上を代入した stats を players に加える（addStatCollectors)。

「寿命で死ぬ時に人脈に与えた富の平均」を計算する場合は状況が少し異なる。データを集計する時が，毎期ではなく，死ぬ時だけである。この場合，データを集計するコードはプレイヤーを記述するファイルに書くため（後述する），モデルを記述するファイルには（無条件で集計するため Cond が入らない）StatCollectorCSA のインスタンスを名前（ここでは）「Give in Die」と false を引数として生成し，players に加える。

グラフで表示する部分がコード（C2）である。まず，ChartView を chart に作成し，それを players Scape に加え，chart に上述した表示すべきデータを指定する。例えば，「C Ref.」の個数（人数）を表示したい場合は，「Count

```
//コード（C2）（PDGame.java）
chart = new ChartView();
players.addView(chart);
chart.addSeries("Count C Ref.", Color.green);
//...
chart.addSeries("Count D Def.", Color.red);

chart = new ChartView();
players.addView(chart);
chart.addSeries("Average Ref. Wealth", Color.green);
//...

chart = new ChartView();
players.addView(chart);
chart.addSeries("Average Give in Die", Color.red);
```

C Ref.」と Count を付け，2つ目の引数にグラフの色（Color.green）を指定する。また，データ「Ref. Wealth」の平均を表示したい場合は，「Average」を前に付け，第1引数を「Average Ref. Wealth」とする。

プレイヤーが死ぬ時にデータ「Give in Die」を収集するので，この部分はプレイヤーを記述する PDGamePlayer クラスの die() メソッド内で行う（コード5を参照）。

```
//コード 5（PDGamePlayer.java）
public void die() {
    super.die();
    if (((PDGame) scape.getRoot()).takeWealthFromConnections) {
        if (((PDGame) scape.getRoot()).giveWealthWhenDie) {
            if (type == PDGame.REFERENTIAL || type == PDGame.COOPERATOR) {
                int tmpwealth = wealth;
                boolean res = connections.doGive();//人脈に富を与えれば真となる。
                if (res) {
                    scape.getData().getStatCollector("Give in Die")
                                        .addValue(tmpwealth - wealth);
                }
            }
        }
//...
}
```

4　プレイヤーの描画方法を変更する

何もしなければ2次元格子上で表示されるプレイヤーは周と内部が塗られた円形である。この色は getColor() メソッドで指定できる。しかし，例えば，

タグの個数が 4 個の場合，C を取っている Ref.，D を取っている Ref.， Coop.， Def. を見分けるためには16通りの区別をする必要があるが，これを色の違いだけですることは困難である。Ref. は円形，Coop. は正方形，Def. は三角形と形を変える。タグに関しても（4 種類の）区別をする（表 4 参照）。

表 4

tag	0	1	2	3
Ref. C				
Ref. D				
Coop.				
Def.				

このようにプレイヤーの形を変更するにはもう少し手間がかかる。変更後のプレイヤーの描画方法を記述した DrawFeature クラスの定数 DRAW_PD_PLAYER を定義する（コード 6 を参照）。draw 関数の obj に描画するセルが入っており，プレイヤーがいる場合に，幅 width，高さ height の中に Graphics g を利用して描く。この定数はプレイヤーを記述する PDGamePlayer クラスで定義する。

```
//コード 6（PDGamePlayer.java）
public final static DrawFeature DRAW_PD_PLAYER = new DrawFeature("PD Player") {

    public void draw(Graphics g, Object obj, int width, int height) {
        PDGamePlayer player = (PDGamePlayer) ((HostCell) obj).getOccupant();
        if (player != null) {
            Graphics2D g2 = (Graphics2D) g;
            g2.setStroke(new BasicStroke(1.5f));
            g2.setColor(Color.black);
            switch (player.type) {
                case PDGame.REFERENTIAL: // circle
                    if (player.move == PDGame.MOVE_C) {
                        g2.draw(new Ellipse2D.Double(1, 1, width - 1, height - 1));
                    } else {
                        g2.fill(new Ellipse2D.Double(1, 1, width - 1, height - 1));
                    }
                    break;
                case PDGame.COOPERATOR:   // square
                    g2.draw(new Rectangle2D.Double(1, 1, width - 1, height - 1));
                    break;
                case PDGame.DEFECTOR:   // triangle
                    int xPoints[] = {width / 2, 1, width - 1};
                    int yPoints[] = {1, width - 1, width - 1};
                    GeneralPath polygon = new
                      GeneralPath(GeneralPath.WIND_EVEN_ODD, xPoints.length);
                    polygon.moveTo(xPoints[0], yPoints[0]);
                    for (int index = 1; index < xPoints.length; index++) {
                        polygon.lineTo(xPoints[index], yPoints[index]);
                    }
```

```
                    polygon.closePath();
                    g2.fill(polygon);
                    break;
                }

                // tag
                int xPoints[] = {width / 2, width / 2, 2, width - 2};
                int yPoints[] = {2, height - 2, height / 2, height / 2};
                GeneralPath polygon = new
                        GeneralPath(GeneralPath.WIND_EVEN_ODD, xPoints.length);
                polygon.moveTo(xPoints[0], yPoints[0]);
                for (int index = 1; index < Math.min(player.tag + 1,
                                                        xPoints.length); index++) {
                    polygon.lineTo(xPoints[index], yPoints[index]);
                }
                if (player.move == PDGame.MOVE_C) {
                    g2.setColor(Color.black);
                } else {
                    g2.setColor(Color.white);
                }
                g2.setStroke(new BasicStroke(0.0f));
                g2.draw(polygon);
            }
        }
    };
```

この定数 DRAW_PD_PLAYER を，モデルを記述する PDGame クラスで lattice に加え（addDrawFeature），必要な設定を行う（コード（C3）を参照）。この「コード（C3）」はコード 1 の該当部分に挿入する。

```
//コード（C3）
overheadView.setDrawByFeature(true);
lattice.addDrawFeature(PDGamePlayer.DRAW_PD_PLAYER);
overheadView.getDrawSelection().clearSelection();
overheadView.getDrawSelection().setSelected(PDGamePlayer.DRAW_PD_PLAYER, true);
overheadView.getDrawSelection().setSelected(overheadView.cells_fill_draw_feature, true);
overheadView.getDrawSelection().moveToFront(overheadView.cells_fill_draw_feature);
```

5 Sweep

シミュレーションで定性的，定量的分析を行うためには，（結果の画面表示を行わず；view＝false）パラメータをある範囲で変化させて（いわゆる，sweep），指定された期間数（stopPeriod）のシミュレーションを多数回（runsPer）行い，結果をファイル（runFile＝true，periodFile＝true）に保

存することが必要である。また，上記のデータの中から興味のあるケースを選びその乱数種（randomSeed）を指定して，今度は，結果の画面表示を行って（view＝true），シミュレーションを行うことも必要である。これを NetBeans IDE 上で，上記の変数（及び，必要ならば他のパラメータの初期値）を指定することによって行いたい。まず，sweep において変化させるパラメータの set メソッドと get メソッドを記述しておく必要がある。

このことを行うために PDGame クラスの派生クラス PDGameMultiple（コード 7 を参照）を作成する。コード 7 の Part（a）の部分で，結果の画面表示を行うか（view＝true），期間数（stopPeriod），結果をファイルに保存するか（runFile＝true，periodFile＝true），シミュレーションの実行回数（runsPer），等を指定する。Part（b）では sweep を行わない他のパラメータの初期値を設定する。PDGame クラスの変数宣言部分で既定値を設定しているが，それを変更する場合に，この Part（b）の部分で初期化を行う。以上の部分はモデルを記述するクラスの createScape() の中で実行される。

```
//コード 7（PDGameMultiple.java）
public class PDGameMultiple extends PDGame {

    boolean view = false;
    boolean runFile = false;
    boolean periodFile = false;
    long randomSeed = ARBITRARY_SEED;
    int runsPer = 30;
    int stopPeriod = 500;
    int pausePeriod = 320;

    /*
     * myCreateScape で初期値を設定。
     * mySweep で sweep を設定
     */
    private void myCreateScape() {
        // Part (a)
        //view = true;   // set true if you want to watch the graphic views
        //pausePeriod = 320;   // set pausePeriod (default value = 320) if view = true

        //stopPeriod = 500;   // set the number of periods (default value = 500)
        //runFile = true;   // set true if you want to output run file
        //periodFile = true;   // set true if you want to output period file

        //runsPer = 1;   // set the number of runs (default value = 30)
        //randomSeed = 1279765355770L;   // set randomSeed if you want to specify randomSeed
```

```
        // Part (b)
        // set initial values of other parameters if you need
        //canTakeFromDifferentTag = false;
        //takeWealthFromConnections = true;
        //giveWealthWhenDie = true;

        if (runsPer == 1 ) {
            setAutoRestart(false);
        }

        if (runsPer == 1 && view) {
            setPausePeriod(pausePeriod);
            super.pause = true;
        } else {
            try {
                setStopPeriod(stopPeriod);
            } catch (SpatialTemporalException e) {
                System.out.println(e);
            }
        }

    }

    private void mySweep() {
        Date today;
        String output;
        SimpleDateFormat formatter;
        String pattern = "yyyyMMddHHmmss";
        formatter = new SimpleDateFormat(pattern);
        today = new Date();
        output = formatter.format(today);
        DataOutputView outputView = new DataOutputView();

        String filename = "PDGameRun" + output + ".txt";
        String filename1 = "PDGamePeriod" + output + ".txt";

        try {
            if (runFile) {
                outputView.setRunFile(new File(filename));
            }
            if (periodFile) {
                outputView.setPeriodFile(new File(filename1));
            }
        } catch (Exception e) {
            System.out.println(e);
        }
        addView(outputView);

        SweepControlView sweeper = new SweepControlView();
        SweepLink varDim = new SweepLink();
        // Part (c)
        // repeat the following as many times as you need to set sweep variables
        varDim.addMember(new SweepDimension(this, "numberOfTags", 0, 4, 1));
        sweeper.getSweepGroup().addMember(varDim);

        sweeper.getSweepGroup().setRunsPer(runsPer);
        addView(sweeper);

    }

    public void createScape() {
```

```
            myCreateScape();
            super.createScape();
        }

        public void createGraphicViews() {
            mySweep();
            if (view) {
                super.createGraphicViews();
            }
        }

        public void scapeSetup(ScapeEvent scapeEvent) {
            super.scapeSetup(scapeEvent);

            setRandomSeed(randomSeed);
        }

        public void scapeInitialized(ScapeEvent se) {
            super.scapeInitialized(se);
            if (view) {
                getRunner().pause();
            }
        }

    }
```

結果をファイルに保存する場合は，DataOutputView を作り，出力ファイルを指定し，この outputView を Scape に加える（addView)。sweep を行うためには Part（c）の部分のように行う。ここでは，numberOfTags の値を 0 から始め 4 を超えるまで増分 1 ずつで実行する。注意する点は numberOfTags の綴りを間違えないことである。このコード部分 mySweep() は PDGameMultiple クラスの createGraphicViews() の中で実行される。もし，view が偽ならば，PDGame クラスの createGraphicViews() は実行されず結果の画面表示がされない。

乱数種の設定はモデルを記述するクラスの scapeSetup(ScapeEvent) メソッドの中で行う。

Ascape の現在の仕様では，モデルを実行するとすぐにシミュレーションも開始される。Ascape 起動時にはロゴなどが前面に表示されるため，このままではシミュレーションの最初の部分が見えない。これを回避するために，scapeInitialized(ScapeEvent) 内で view が真ならば，第 0 期の時にシミュレーションの実行を一時停止させる。これにより，初期状態をユーザーが確認できる。

6 そ の 他

幾つかの留意点を述べる。モデル PDGame では Ref. と Coop. がプレイヤーの中からいなくなった時点で，（突然変異を無視すれば協調行動が今後起こらないという意味で）以後のシミュレーションを行う必要がない。これをチェックしてシミュレーションを止める部分が，PDGame クラスの（繰返し毎に呼び出される）scapeIterated(ScapeEvent）メソッド内のコードの部分である（コード 8 を参照)。コード（C1）の部分で作成している StatCollector で既に計数しているので，それを参照して Ref. と Coop. の人数が 0 以下になれば，シミュレーションを停止する。コード内で pause が真の場合（これは PDGameMultiple が実行されて，結果の画面を表示させる場合である)，stop() を実行すると結果の画面が消えてしまうので，これを避けるため pause() を実行している。

```
                                    //コード 8（PDGame.java）
public void scapeIterated(ScapeEvent scapeEvent) {
      super.scapeIterated(scapeEvent);
      DataSeries dataSeriesCountCRef = getData().getSeries("Count C Ref.");
      DataSeries dataSeriesCountDRef = getData().getSeries("Count D Ref.");
      DataSeries dataSeriesCountCCoop = getData().getSeries("Count C Coop.");
      double countRefCoop = dataSeriesCountCRef.getValue()
                  + dataSeriesCountDRef.getValue() + dataSeriesCountCCoop.getValue();
       Boolean cond = countRefCoop <= 0;
       if (cond) {
           if (pause) {
                getRunner().pause();
           } else {
                getRunner().stop();
           }
      }

}
```

図 1 のウィンドウの上部のコントロールボタンの右から 9 個目を押すと，図 2 の Setting Window が現れる。左側が Parameters タブ，右側が Rules タブが選ばれた時の様子である。左側の Parameters タブで，Value の欄を修正することによって，シミュレーション実行中にパラメータの値を変更できる。右側の Rules タブではルールを無効化したり，順序を変えることができる。

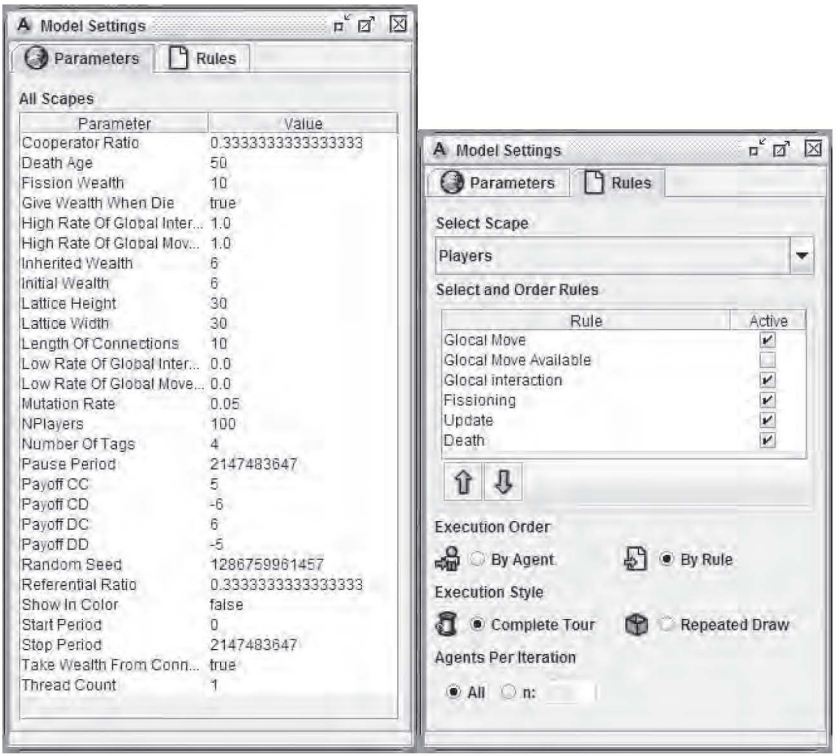

図 2　Setting Window

図 1 のウィンドウの上部のコントロールボタンの右から 5 個目の Information ボタンはモデルの情報を表示させるボタンである。これを実際に作動させるためには，AboutPDGame.html という名前のファイルに〈html〉タグ，〈head〉タグ，〈body〉タグを使用せずに，直接内容を書き，（PDGame.java と同じディレクトリに）保存する。このことにより（NetBeans では）jar ファイルの適切な場所に格納される。MyAscapeModels.jar を実行し，Information ボタンを押すと，図 3 のようにウィンドウが現れてその中に AboutPDGame.html に記述した内容が表示される。

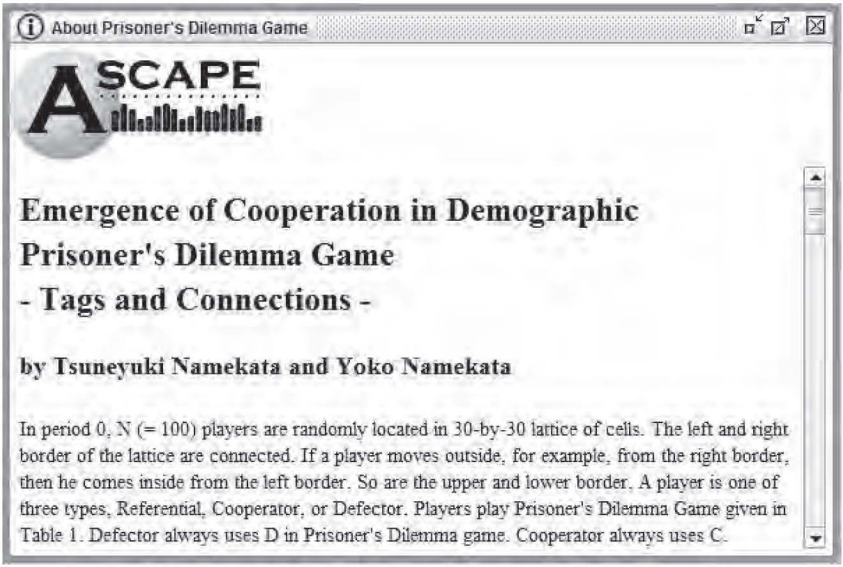

図3　モデルの情報

Ascape の既定では2次元格子上のセルの色は緑色である。これを変更する一番簡単な方法はコード1の MyHostCell() の部分を無名の内部クラス HostCell を利用し，その中で getColor() を定義し，自分の好きな色を返せばよい。しかし，このように行うと，図1の2次元格子上のプレイヤーをポイントしてプレイヤーの状態を調べる（Alt+Click），またはプレイヤーを追跡する（Shift+Alt+Click），を試みた場合，（原因は不明だが）Ascape の Log ウィンドウに下記のようなエラーメッセージが表示される：

Error in dynamic method read: java.lang.IllegalAccessException:
Class org.ascape.util.PropertyAccessor can not access a member of class org.namekata.ascape.models.pd.PDGame$1 with modifiers "public"

これを回避するために MyHostCell クラスを（HostCell クラスの派生クラスとして）定義し，その中の getColor() メソッドで好きな色を返す。

7 アプレット

```
//アプレットタグ
<applet width="100%" height="100%"
      code="org.ascape.runtime.applet.SwingApplet"
      archive="MyAscapeModels.jar,Ascape_Applet5.6.0.jar">
          <param name="Model" value="org.namekata.ascape.models.pd.PDGame"/>
</applet>
```

アプリケーションとして作成した MyAscapeModels.jar を利用してアプレット上でモデルを実行することは容易である。アプレット上で実行するために Ascape_Applet5.6.0.jar が準備されている。PDGame.html という html ファイルを作成し，（多くの幅を必要とするので適切な場所に）「アプレットタグ」を挿入する。PDGame.html のあるディレクトリに MyAscapeModels.jar と Ascape_Applet5.6.0.jar をコピーし，PDGame.html をブラウザで実行し，ブラウザの幅を大きくし中の幅を適宜修正すると，図 4 のようになる。

コントロールボタンの数は減っているが，アプレケーションとして作成したモデルをそのまま利用してアプレットとして表示できるので大変助かる。

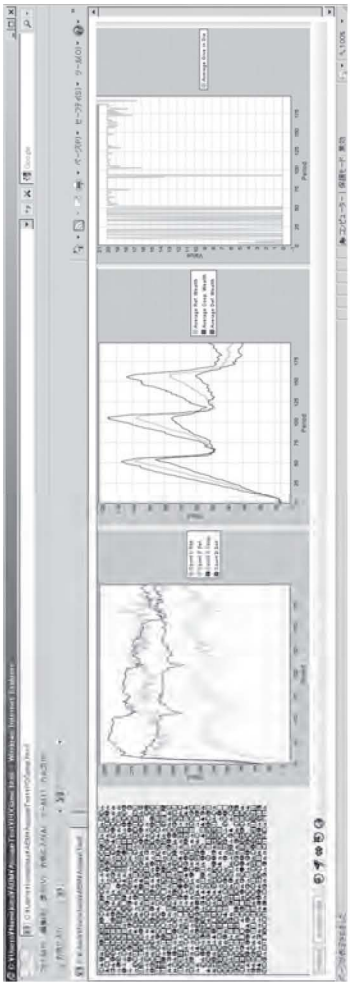

図 4 アプレットとして実行

5．終わりに

本稿では 2 次元格子上で囚人のジレン

マゲームを行う文献[3]のモデルを例に取り，エージェントベースモデルの開発・探索ツールである Ascape の利用方法を説明した。

IDE として，Eclipse ではなく NetBeans で利用するために若干の工夫を要する部分，sweep の方法，プレイヤーの描画方法を変える方法，等，マニュアル[1]，[2]には載っていない部分を中心に実装方法を述べた。

6．補　　遺

ソースファイルの抽出：

Ascape_SDK_5.6.0.zip からソースファイルを抽出する。具体的には Ascape_SDK_5.6.0.zip を unzip する。作成されたディレクトリ Ascape_SDK_5.6.0にある org.ascape.core や org.ascape.ui.swing 等のディレクトリ内の src ディレクトリの内容を（例えば，新規に作成した）Rev_Ascape_Source_5.6.0ディレクトリ内にコピーする。Rev_Ascape_Source_5.6.0ディレクトリには org，edu，name ディレクトリができ，Ascape_SDK_5.6.0.zip 内のソースファイルだけがコピーされたことになる。

ModelChoices.txt の修正：

プロジェクトウィンドウの MyAscapeModels を右クリックして［新規］-［Java クラス...］を選び，パッケージ：の右に「org.ascape.model」と入力し，（不要なので後で削除するためクラス名はそのままにして）完了ボタンを押す。(作成されたクラスファイルは不要なので削除しておく。) 同様に，［新規］-［その他］を選び，表示されるダイアログボックスで「その他」，「空のファイル」を選び，次へボタンを押す。フォルダを今作成した org.ascape.model

//ModelChoices.txt の内容

```
Models
{Examples
    PDGame|org.namekata.ascape.models.pd.PDGame
    PDGameMultiple|org.namekata.ascape.models.pd.PDGameMultiple
}
```

に設定し，名前を「ModelChoices.txt」にし完了ボタンを押す。このファイルに「ModelChoices.txt の内容」を記述して MyAscapeModels.jar を作成すれば，Ascape_5.6.0.jar にあるものよりも優先され，リストから Example を選べば，PDGame と PDGameMultiple のみが表示される。

参考文献

[1] Ascape Guide (http://ascape.sourceforge.net/)

[2] The Ascape Model Developer's Manual (http://ascape.sourceforge.net/manual/Overview.html)

[3] Tsuneyuki Namekata and Yoko Namekata. 2010. "Emergence of Cooperation in Demographic Prisoner's Dilemma Game − Tags and Connections −." In T. Itoh and K. Suzuki, eds. *Proceedings of the 13th Czech-Japan Seminar on Data Analysis and Decision Making in Service Science, November 3-5, Otaru,* 149-154, 2010.