卒業論文

三角格子上の
ジャンケン・シミュレーション

西内啓之

青山学院大学・理工学部・物理学科
羽田野研究室

２００２年度

三角格子上の
ジャンケン・シミュレーション

西内啓之*
羽田野研究室

平成15年1月20日

概 要

普段の生活において、社会の利益は全く考えずに、それぞれが自分の利益だけを求めて行動するという場面がしばしばある。そのとき社会全体はどのように振る舞うのか、その様子を調べるのが本研究の目的である。

人間は各自の考えに基づいて行動する。そこで、ゲーム理論の戦略を用いて三角格子上でジャンケンを行うモデルを導入した。すると、微視的には見られなかった巨視的な現象である渦と吸い込みが見えた。

*青山学院大学 理工学部 物理学科

目 次

1　はじめに

普段の生活において、人間は社会の利益など全く考えずに、自分の利益だけを求めて行動しがちである。それぞれが自分の利益だけを求めて行動した場合、どのような社会になるのか。これを解明するのが本研究の目的である。

人間の行動パターンは、様々である。利益を求めようとする場合においても、様々な行動が考えられる：

- 何も考えずに行動する（ランダム戦略)。
- ひねくれて行動する（天の邪鬼戦略)。
- 単調に行動する（スピン戦略・バックスピン戦略)。
- 成功者を真似て行動する（しっぺ返し戦略)。
- 条件反射に行動する（パブロフ戦略)。
- みんなと同じ行動する（多数派戦略)。
- みんなと逆の行動する（少数派戦略)。

このように、それぞれが戦略を持って行動し、相手の取る戦略が、自分の次の戦略に影響を及ぼす場面を分析対象とするのがゲーム理論である[1, 2, 3, 4]。

本研究では、多くの人がしっぺ返し戦略を採用したとき、どのようなダイナミクスになるのかを調べる。しっぺ返し戦略は、ゲーム理論の代表的な例の一つである、囚人のジレンマの最善解である。また、誰もが成功者を真似たいと思うのは、ごく自然なことである。しっぺ返し戦略だけの社会（しっぺ返し社会）は、どのようなものかを知るため、この点に絞ってモデル化を試みた。

第 2 節では、本研究で扱うモデルの説明をする。三角格子上に配置されたプレーヤーが、繰り返しジャンケンをするというモデルである。第 3 節では、モデルをシミュレーションした結果について報告する。渦と吸い込みというダイナミクスが得られ、その仕組みなどについて説明する。ここまでは、全プレーヤーがしっぺ返し戦略を用いる場合についてだが、第 4 節では、しっぺ返し社会にランダム戦略を混ぜたときの様子について報告する。

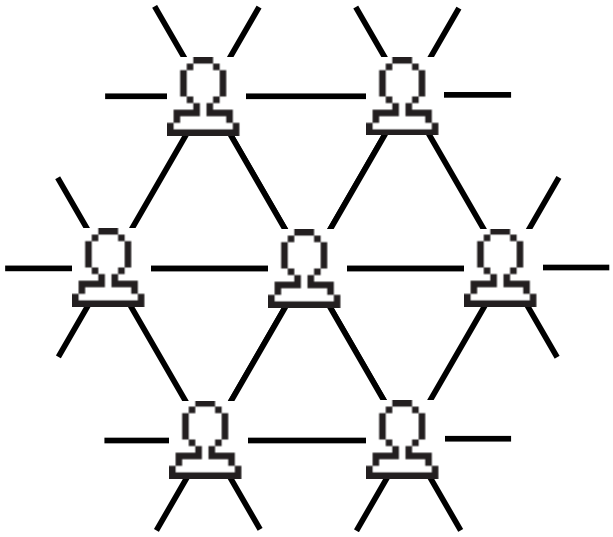

図 1: 三角格子上に並んだプレーヤー。

2　モデルの説明

この節では、今回考えたモデルの説明をする。モデルとしては、三角格子上に配置されたプレーヤー間のジャンケンを扱う。プレーヤーはそれぞれ、しっぺ返し戦略で次の手を決める、というダイナミクスのモデルである。

三角格子の各格子点にプレーヤーがいるとする。一斉にジャンケンの手を出し、自分の周りにいる六人のプレーヤーと一斉に勝負を着ける（図1)。それぞれ得点を、勝ったとき 1 点、あいこで 0 点、負けると −1 点とする。全プレーヤーの得点の合計は常に 0 点となり、全体の利益は一定である（ゼロ和ゲーム)。

以上の操作を何度も繰り返す。その際、全プレーヤーに、しっぺ返し戦略を持たせる。ここでのしっぺ返し戦略とは、周りの六人のプレーヤーの中で最高得点を記録したプレーヤーが出した手を、次回に真似るというものである。最高得点者が複数いる場合は、その中からランダムに選ぶことにする。

この他に、以下のような戦略も考慮する：

- ランダム戦略
  毎回、ランダムに手を選ぶ。

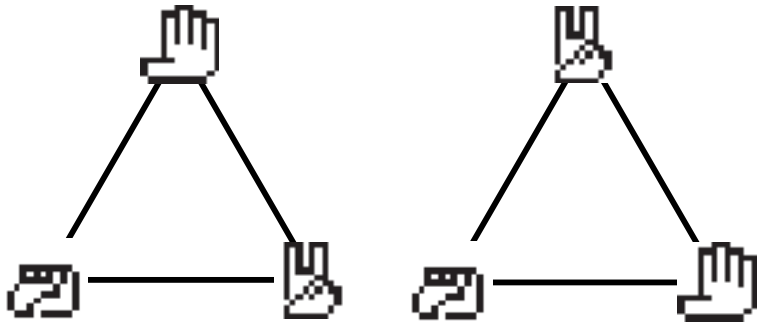

図 2: 三角格子上のフラストレーション。

- 天の邪鬼戦略
  前回とは違う手を選ぶ。
- スピン戦略
  グー→チョキ→パーの順に手を選ぶ。
- バックスピン戦略
  スピン戦略とは逆、パー→チョキ→グーの順に手を選ぶ。
- パブロフ戦略
  前回の得点が、正なら同じ手、負なら違う手、ゼロならランダムに手を選ぶ。
- 多数派戦略
  前回、周りの六人のプレーヤーの中で、最も多い手を選ぶ。多い手が複数ある場合は、その中からランダムに選ぶ。
- 少数派戦略
  前回、周りの六人のプレーヤーの中で、最も少ない手を選ぶ。少ない手が複数ある場合は、その中からランダムに選ぶ。

三角格子上では、グー・チョキ・パーの三つ巴状態（フラストレーション）が起こり得る（図2)。このため、三角格子では、正方格子などより、複雑な振る舞いをすることが期待できる。なお、以下ではプログラムの都合上、三角格子を図3のように変形して、正方格子として表現する。

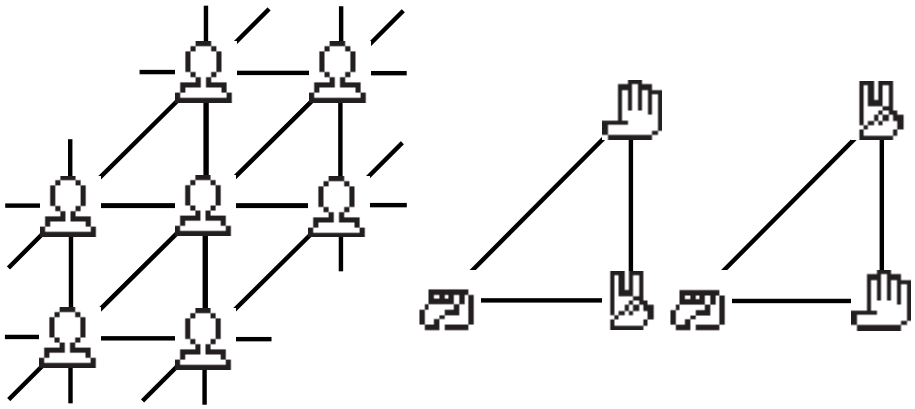

図 3: 三角格子を上のように歪めて、正方格子として表現する。

3　モデルの結果

3.1　シミュレーション

この節では、シミュレーション結果について報告する。渦と吸い込みという、巨視的な現象が存在することが解った。

以下では、主に 64×64 の格子上のシミュレーション結果を示す。なお、ここでは、青がグー、赤がチョキ、黄がパーを表す。上下左右には、周期的境界条件を課しており、初期条件は、それぞれにランダムに手を与えた（図 4）。

シミュレーションをした結果、初めの段階では、同じ手同士が固まっていく様子が見えた（図 4 →図 5 →図 6 →図 7）。しばらくすると、渦を巻いたり（渦）、中心部に吸い込んだり（吸い込み）する現象が見られた（図 8 →図 9 →図 10 →図 11）。渦には、左回りのもの（図 12）と、右回りのもの（図 13）がある。この二つの渦の中心が、互いに強く影響を及ぼし合う距離にまで近付くと、互いに打ち消し合う。一方、吸い込み（図 14）と逆の湧き出しは存在しない。渦・吸い込みの仕組みについては、第 3.4 節で説明する。

極限はほぼ周期的なリミットサイクルとなった。ただし、次の手を決める際に、わずかながら乱数の項を含んでいる（周囲に最高得点者が複数いる場合は、その中からランダムに真似る相手を選ぶ）。従って、完全なリミットサイクルにはならず、少しばかり揺らいでいる。

プレーヤーの数を 256×256 にしたところ、渦は大きくならずに、渦の数が増える結果となった（図 15）。このことから、渦と吸い込みを議論するのに、64×64 の格子で大きさが十分であることが解る。

プレーヤーの数が 64×64 と 256×256、どちらの場合も、渦と吸い込みの厚みは、ほとんどが三段で構成されていた（図 16）。これは、隣の手を侵食する段、自分の手を守る段、隣の手から侵食される段に仕事を分担するのが安定であるためと考えられる。

3.2　定常状態への収束

この節では、系の定常状態への収束について説明する。系が収束したかどうかは、左回りのフラストレーションの数から判断した。図 17 より、だいたい 100 ステップあたりで落ち着くので、100 ステップで定常状態に収束するとした。ここで、左回りのフラストレーションとは図 2 の左側

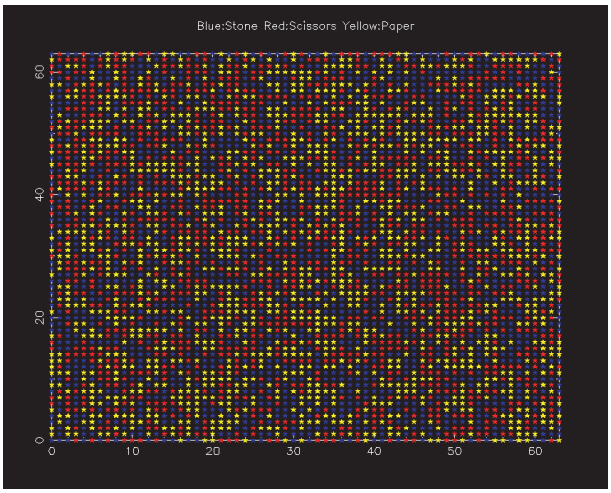

図 4: 初期状態の分布はランダムに与える。

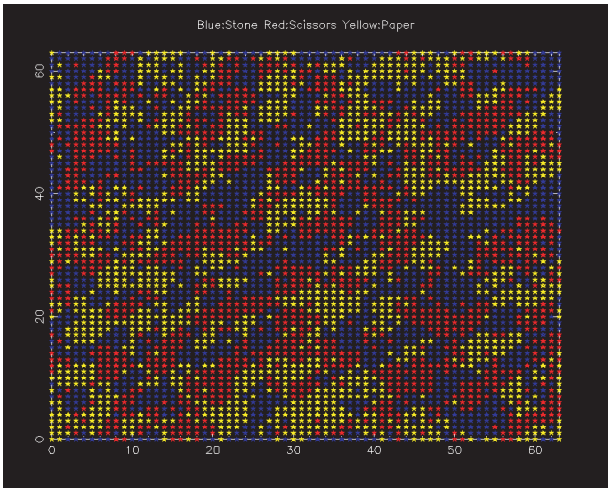

図 5: スナップショット（1 ステップ目）。

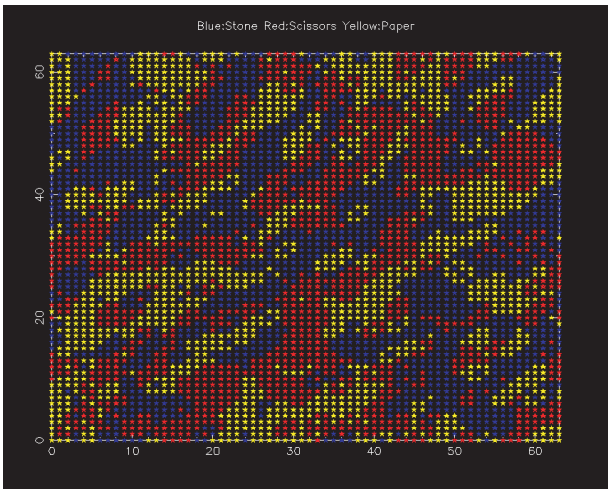

図 6: スナップショット（2 ステップ目）。

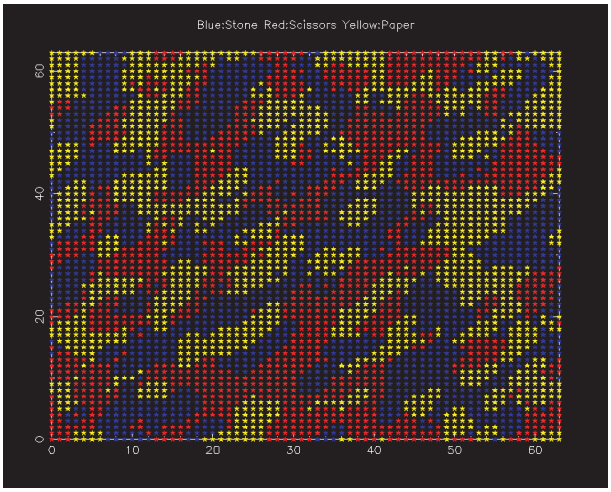

図 7: スナップショット（3 ステップ目）。

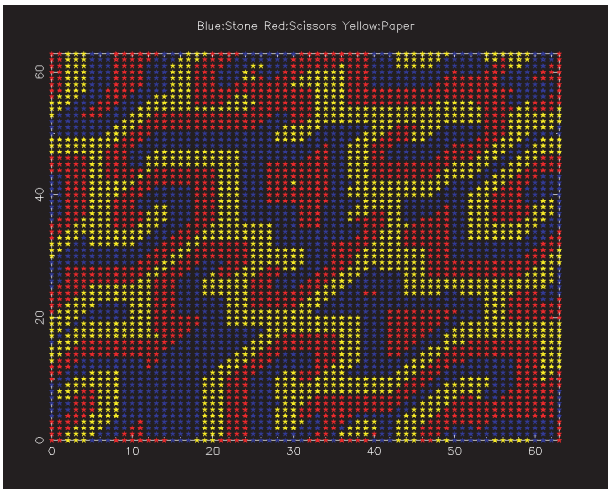

図 8: スナップショット（1020 ステップ目）。

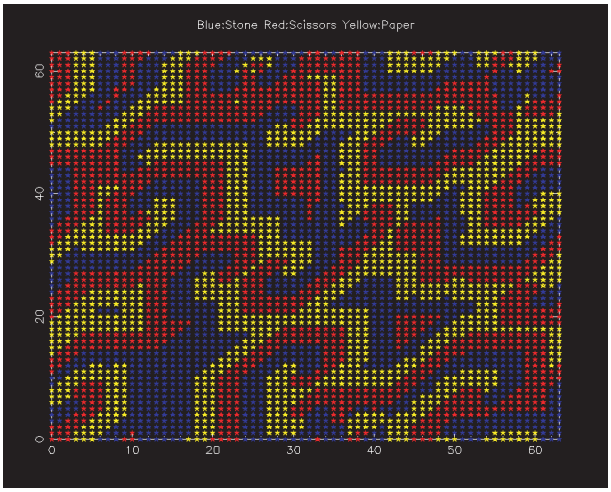

図 9: スナップショット（1021 ステップ目）。

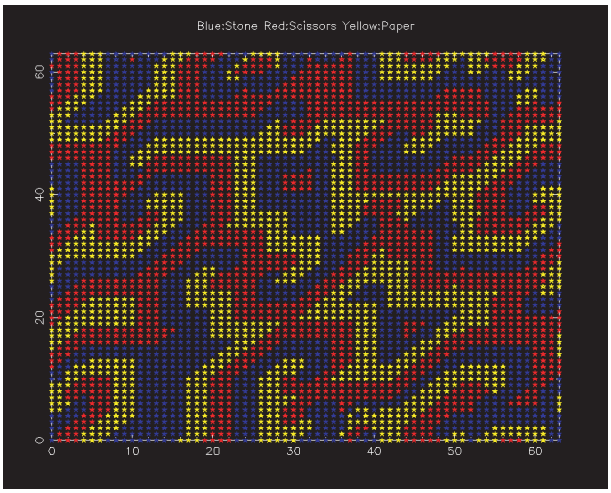

図 10: スナップショット（1022 ステップ目）。

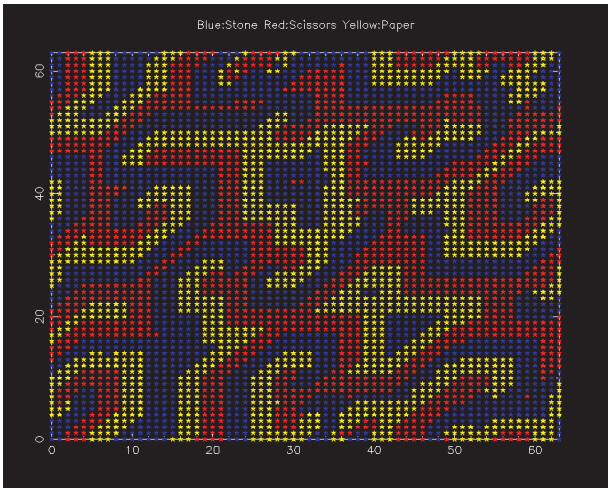

図 11: スナップショット（1023 ステップ目）。

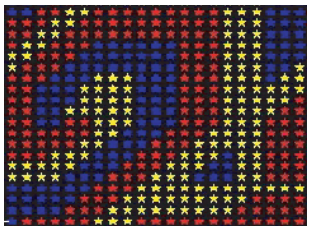

図 12: 左回りの渦。

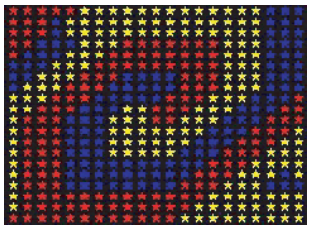

図 13: 右回りの渦。

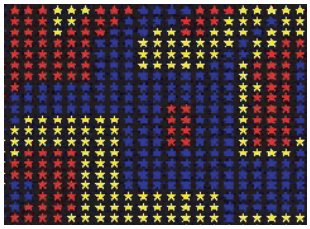

図 14: 吸い込み口。

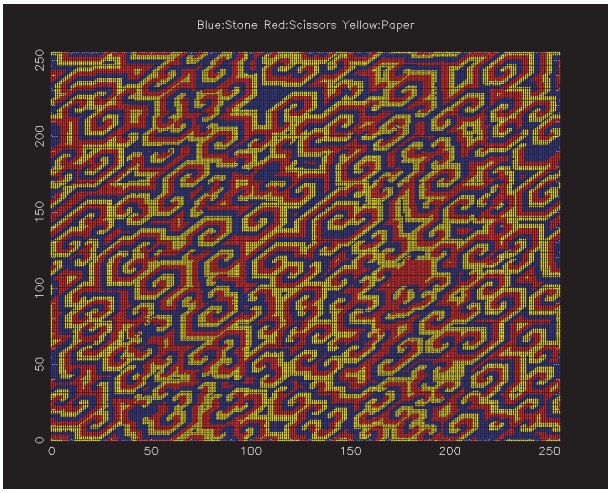

図 15: 256 × 256 の三角格子上のシミュレーションのスナップショット（1023 ステップ目）。

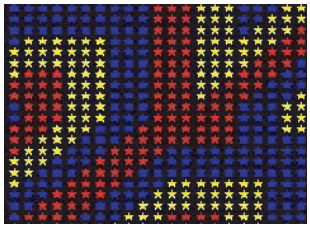

図 16: 同じ手を出すプレーヤーの層が三段で構成されている。

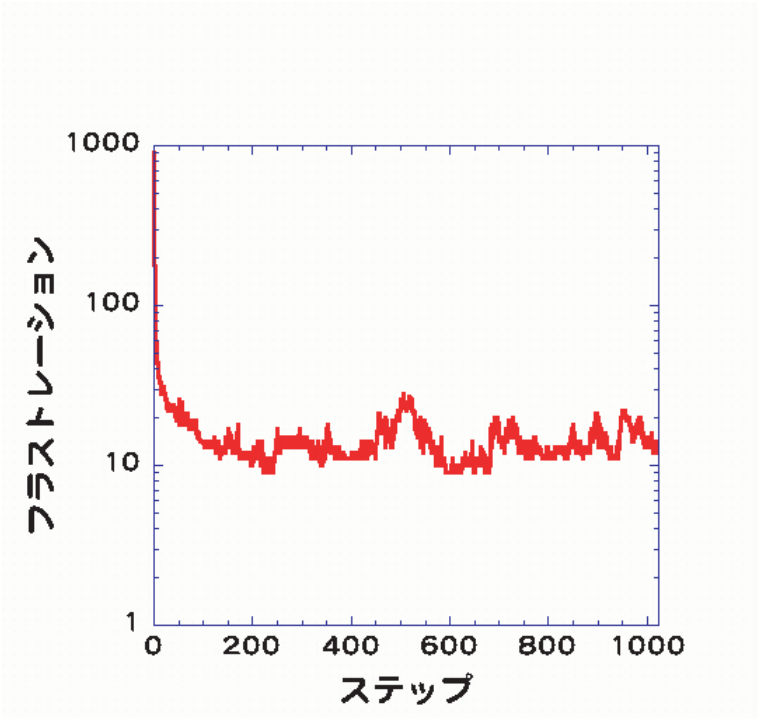

図 17: フラストレーションの数の時間依存性。

のフラストレーション、右回りのフラストレーションとは図 2 の右側のフラストレーションを指す。この二つのフラストレーションは常に同数だけ存在する。

なぜ、この二つは常に同数なのか、その理由を説明する。フラストレーションが起こる場所は図 18 のようなところである。よって、左回りのフラストレーションと右回りのフラストレーションは必ずペアで現れる。よって常に同数になる。

3.3 得点分布

この節では、得点分布について定量的に説明する。1023 ステップ目の得点分布は図 19 のようになった。高得点の部分と低得点の部分に、はっきりと分れているのが解る。一方、図 20 に全員がランダム戦略の場合のシミュレーション結果を示す。図 19 と図 20 を比べると、図 19 で得点の分布に差があるのが、より一層明らかである。つまり、しっぺ返し戦略

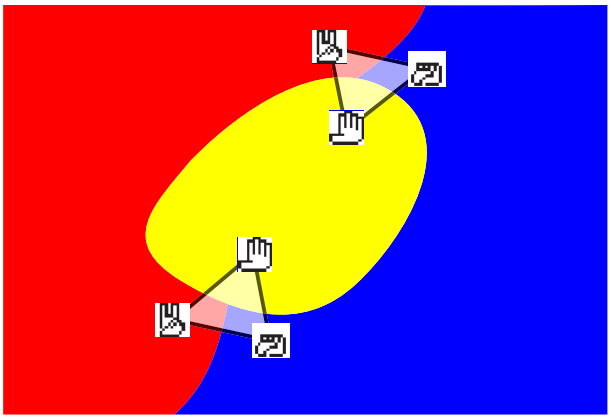

図 18: フラストレーションの発生場所。

	標準偏差
しっぺ返し戦略	0.123
ランダム戦略	0.063

表 1: 全プレーヤーがしっぺ返し戦略を採った場合と、ランダム戦略を採った場合の、1023 ステップ目の全プレーヤーの得点の標準偏差。

は、ランダム戦略に比べ、得点の分布に幅がある。プレーヤーの得点の標準偏差を計算したところ、表 1 のようになった。このことから、しっぺ返し戦略はランダム戦略の倍近く、得点の貧富に差が生じることが解る。

図 11 と図 19 を見比べると、高得点のところでは渦、低得点のところでは吸い込みになっている。この理由については、第 3.4 節で説明する。また、渦と吸い込みは絶えず動いているので、多くのステップにわたって平均すると得点の標準偏差はゼロに収束してしまう。よって以降では、ステップ数を 1023 に固定して議論する。

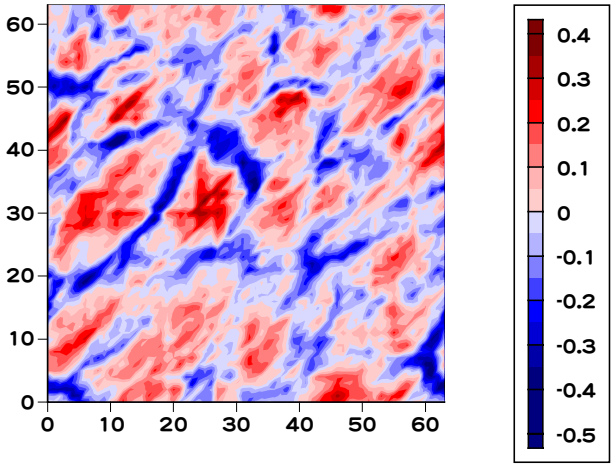

図 19: しっぺ返し戦略の場合の得点分布（1023 ステップ目）。

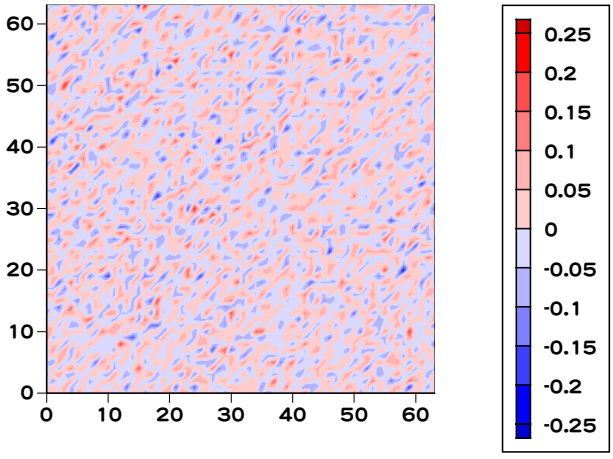

図 20: ランダム戦略の場合の得点分布（1023 ステップ目）。

3.4　渦と吸い込み

この節では、渦と吸い込みの仕組みについて、定性的に説明する。渦と吸い込みでは、境界部分においてのみ、勝負が着く。境界部分では、しっぺ返し戦略のため、弱い手は得点の高かった強い手に侵食される。よって、渦は図 21 のように回転し、吸い込みは図 22 のように中心に吸い込まれる。負け続ける吸い込み口以外では、順々に勝ち負けが繰り返される。

それでは、なぜ勝ち負けの数に大差無いにも関わらず、得点に差ができるのだろうか。それは、境界部分が曲がっているからである。境界部分が曲がっていると、勝つプレーヤーの数と負けるプレーヤーの数が同じではない（図 23)。一方で、このゲームはゼロ和ゲームなので、勝つプレーヤーの総得点と負けるプレーヤーの総失点は等しい。よって、得点の分け前に偏りができる。

渦付近では勝つプレーヤーが境界の内側になるので、数が少ない。よって得点の分け前が多くなる。一方、吸い込み付近では勝つプレーヤーが境界の外側になるので数が多い。よって得点の分け前が少なくなる。つまり、渦では高得点、吸い込みでは低得点となる。渦は巨視的なフラストレーションと見ることができるので、フラストレーションがある部分は高得点だと言える。

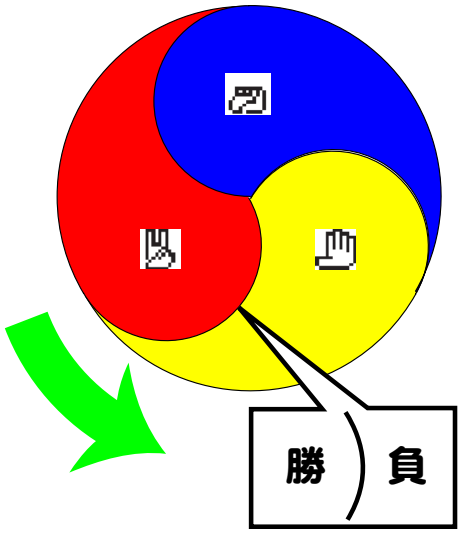

図 21: 渦。

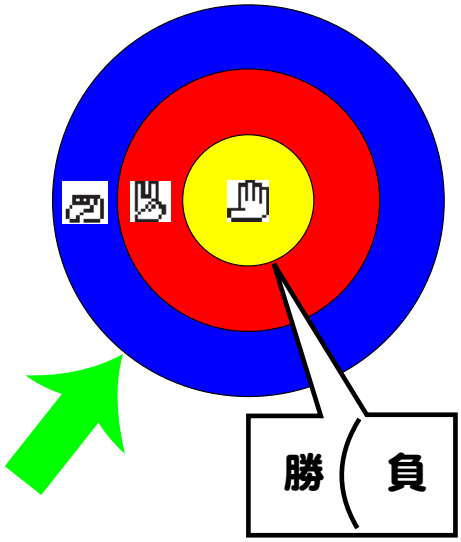

図 22: 吸い込み。

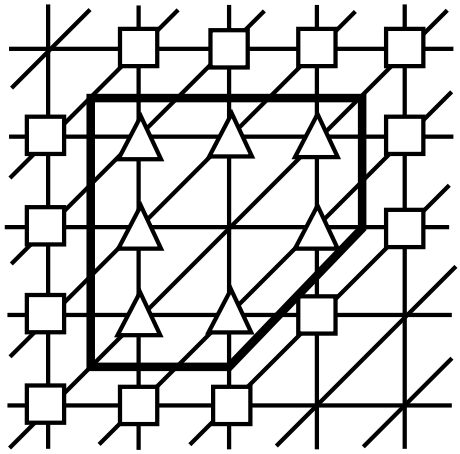

図 23: 境界の例。境界が曲がっているとその内側のプレーヤー (△) の数は外側のプレーヤー (□) の数より少ない。

4　ランダム戦略との混合

この節では、しっぺ返し社会にランダム戦略を混ぜた場合について報告する。一般社会は単純ではない。規律を守る者もいれば、破る者もいる。そこを少しでも再現するため、しっぺ返し戦略にランダム戦略を混ぜていった。

ランダム戦略の混ぜ方には、二通りの方法がある。一つは、初めに各プレーヤーがしっぺ返し戦略か、ランダム戦略かを決めて、それ以降は戦略を変えない場合（戦略固定）である。もう一つは、毎回各プレーヤーが戦略を決める場合（戦略変動）である。戦略固定では、わずか 1%、戦略変動では 10%、ランダム戦略を入れると、渦は無くなり、代わりに湧き出しのような現象（湧き出し（図 24・図 25））が現れた。このことから、渦というのは、ある程度の秩序を必要とする、しっぺ返し戦略特有のものであることが解った。

戦略固定と戦略変動では、得点の標準偏差に図 26 のような違いが現れた。戦略固定では、約 9% で得点の標準偏差が最大（図 27)、戦略変動では、約 4% で得点の標準偏差が最低となった（図 28)。

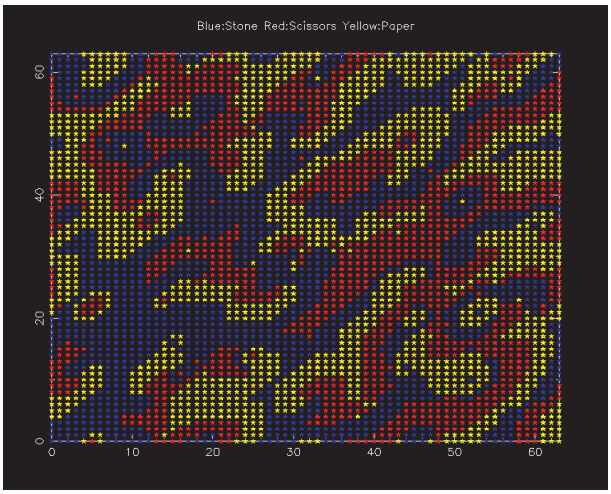

図 24: しっぺ返し戦略が約 99%、ランダム戦略が約 1%、戦略固定で混ざっている場合のスナップショット（1023 ステップ目）。

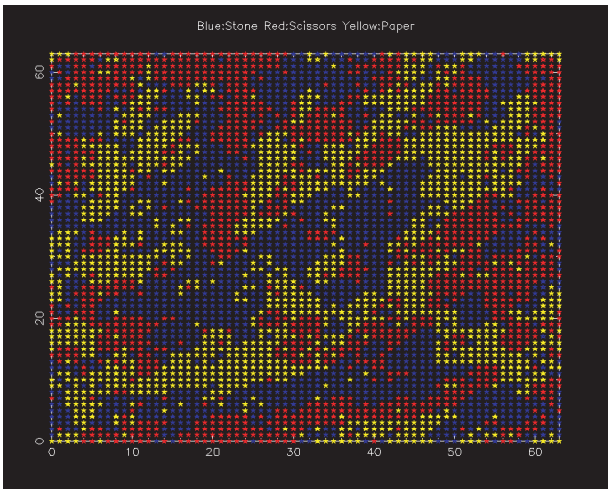

図 25: しっぺ返し戦略が約 90%、ランダム戦略が約 10%、戦略変動で混ざっている場合のスナップショット（1023 ステップ目）。

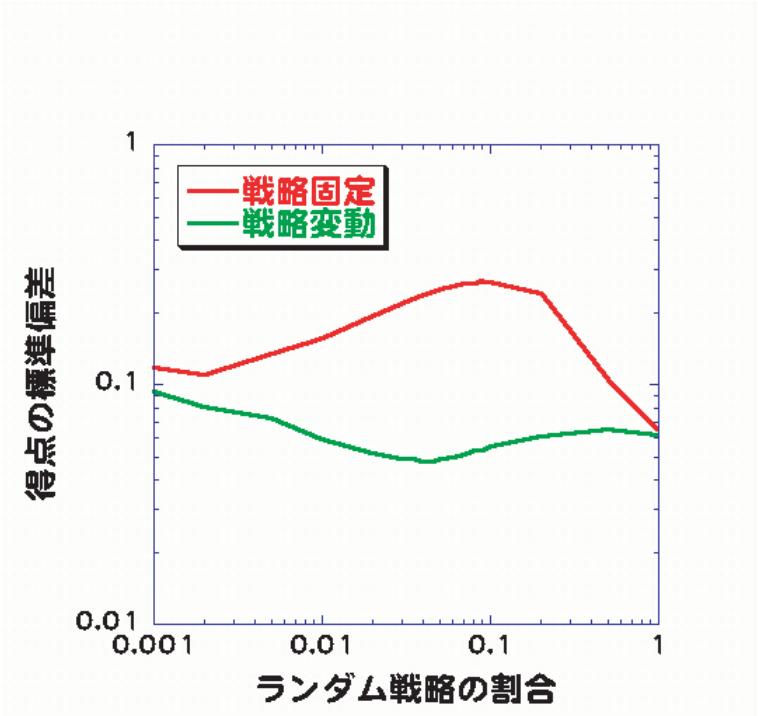

図 26: ランダム戦略の混合割合に対する得点の標準偏差。赤線が戦略固定の場合、緑線が戦略変動の場合。

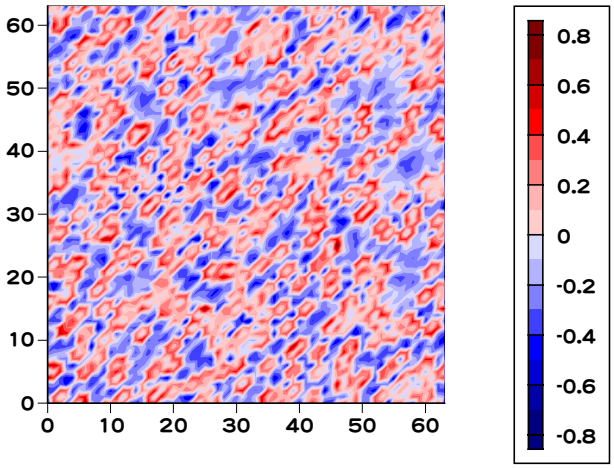

図 27: しっぺ返し戦略が約 91%、ランダム戦略が約 9%、戦略固定で混ざっている場合の得点分布（1023 ステップ目）。

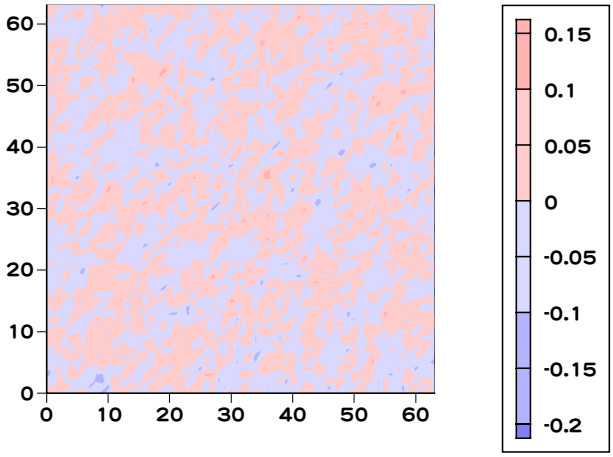

図 28: しっぺ返し戦略が約 96%、ランダム戦略が約 4%、戦略変動で混ざっている場合の得点分布（1023 ステップ目）。

5　まとめ

社会全体の振る舞いを調べるため、三角格子上のジャンケンをモデル化した。シミュレーションした結果、微視的には見られなかった、渦と吸い込みという巨視的な現象が起こった。普段の生活の中で良いものを真似ようとするのは、ごく自然なことなので、条件さえ揃えば、現実社会でも渦と吸い込みが起こる可能性が示せた。

しっぺ返し社会では、ランダム社会より、貧富に差が生じることが解った。より利益を得ようと戦略を持ったのにも関わらず、そのことで、ランダム社会より貧富に差が生じてしまったのは、興味深い結果である。

しっぺ返し社会で高得点を得るには、勝つプレーヤーが少ないところにいればいい。一度、渦の部分に入ってしまえば、あとはほとんど自動的に利益が得られることになる。

しかし、ある程度の純度でないと、しっぺ返し社会の秩序は保たれない。現実社会では、様々な戦略を持ったプレーヤーがいるので、高得点の渦が存在するのは難しいと言える。

参考文献

[1] Manfred Eigen and Ruthild Winkler, “Laws of the Game: How the Principles of Nature Govern Chance” (Princeton Science Library, 1993)

[2] 賀川昭夫 「ゲーム理論」 (東京経済大学, 1999)

[3] Eric D.Demaine, “Playing Games with Algorithms: Algorithmic Combinatorial Game Theory” (Department of Computer Science, 2001)

[4] 船木由喜彦 「エコノミックゲームセオリー 協力ゲームの応用」 (サイエンス社, 2001)

A　三角格子上のジャンケンのプログラムリスト

A.1　プログラムの流れ

(i) 初期状態を設定する。

(ii) 状態を画像で出力する。

(iii) フラストレーションを数える。

(iv) 得点を計算する。

(v) ステップ数だけ (a)〜(d) を繰り返す。

  (a) 戦略により次に出す手を決める。

  (b) 状態を画像で出力する。

  (c) フラストレーションを数える。

  (d) 得点を計算する。

(vi) 総得点を出力する。

A.2 プログラムリスト

```
/*********************************************************/
/*               0:PAPER 1:SCISSORS 2:STONE              */
/*********************************************************/


#include <stdio.h>
#include <stdlib.h>

#define SIMPLE_SPRNG
#include "sprng.h"
#define SEED 19801031

#include "cpgplot.h"

#define _N 64         /* The size of lattices */

#define rate_0 0.000  /* The rate of Natural Strategy */
#define rate_1 0.000  /* The rate of Perverse Strategy */
#define rate_2 0.000  /* The rate of Spin Strategy */
#define rate_3 0.000  /* The rate of Backspin Strategy */
#define rate_4 1.000  /* The rate of TitForTat Strategy */
#define rate_5 0.000  /* The rate of Pavlov Strategy */
#define rate_6 0.000  /* The rate of Majority Strategy */
#define rate_7 0.000  /* The rate of Minority Strategy */


/******** Frustration ********/
int frustration(int first,int second,int third);

/******** Payoff ********/
int payoff(int i,int you);

/******** Periodic boundary conditions ********/
int pbc(int room);
```

```
/******** Natural Strategy ********/
int natural();

/******** Perverse Strategy ********/
int perverse(int cell);

/******** Spin Strategy ********/
int spin(int cell);

/******** Backspin Strategy ********/
int backspin(int cell);

/******** TitForTat Strategy ********/
int titfortat(int mark_0,int pt_0,int mark_1,int pt_1,
              int mark_2,int pt_2,int mark_3,int pt_3,
              int mark_4,int pt_4,int mark_5,int pt_5);

/******** Pavlov Strategy ********/
int pavlov(int cell,int pt);

/******** Majority Strategy ********/
int majority(int mark_0,int mark_1,int mark_2,int mark_3,
             int mark_4,int mark_5);

/******** Minority Strategy ********/
int minority(int mark_0,int mark_1,int mark_2,int mark_3,
             int mark_4,int mark_5);


int main(void)
{
  FILE *FRUSTRATION,*POINT;
  int new[_N][_N],old[_N][_N],pt[_N][_N],pts[_N][_N];
```

```
int number[3];
float character[_N][_N];
float plot_x[3][_N*_N],plot_y[3][_N*_N];
char filename[20];
int frustrations,i=0,j,step,x,y;
init_sprng(DEFAULT_RNG_TYPE,SEED,SPRNG_DEFAULT);

FRUSTRATION=fopen("frustration.dat","w");
POINT=fopen("point.dat","w");

/* Input the step number. */
printf("step=");
scanf("%d",&step);

/* Decide the initial state of the variable. */
for(y=0;y<=_N-1;y++){
  for(x=0;x<=_N-1;x++){
    old[x][y]=(int)(3.0*sprng());
    character[x][y]=sprng();
    pts[x][y]=0;
  }
}

/* Format the variable. */
for(j=0;j<=2;j++){
  number[j]=0;
}
frustrations=0;

/* Prepare PGPLOT. */
for(y=0;y<=_N-1;y++){
  for(x=0;x<=_N-1;x++){
    j=old[x][y];
    plot_x[j][number[j]]=(float)x;
    plot_y[j][number[j]]=(float)y;
```

```
    number[j]++;
  }
}

/******** Perform PGPLOT. ********/

/* Open the file. */
sprintf(filename,"step_%04d.gif/GIF",i);
cpgopen(filename);

/* Make the foundation. */
cpgsci(1);
cpgenv(0.0,(float)(_N-1),0.0,(float)(_N-1),0.0,0.0);
cpglab("","","Blue:Stone Red:Scissors Yellow:Paper");

/* Plot Paper. */
cpgsci(7);
cpgpt(number[0],plot_x[0],plot_y[0],18);

/* Plot Scissors. */
cpgsci(2);
cpgpt(number[1],plot_x[1],plot_y[1],18);

/* Plot Stone. */
cpgsci(4);
cpgpt(number[2],plot_x[2],plot_y[2],18);

/* Close the file. */
cpgclos();

for(y=0;y<=_N-1;y++){
  for(x=0;x<=_N-1;x++){

    /* Count frustrations. */
    frustrations=frustrations
```

```
      +frustration(old[x][y],old[pbc(x+1)][y],
                   old[pbc(x+1)][pbc(y+1)])
      +frustration(old[x][y],old[pbc(x+1)][pbc(y+1)],
                   old[x][pbc(y+1)])
      +frustration(old[x][y],old[x][pbc(y+1)],
                   old[pbc(x-1)][y])
      +frustration(old[x][y],old[pbc(x-1)][y] ,
                   old[pbc(x-1)][pbc(y-1)])
      +frustration(old[x][y],old[pbc(x-1)][pbc(y-1)],
                   old[x][pbc(y-1)])
      +frustration(old[x][y],old[x][pbc(y-1)],
                   old[pbc(x+1)][y]);

    /* Calculate the score. */
    pt[x][y]=payoff(old[x][y],old[pbc(x+1)][y])
      +payoff(old[x][y],old[pbc(x+1)][pbc(y+1)])
      +payoff(old[x][y],old[x][pbc(y+1)])
      +payoff(old[x][y],old[pbc(x-1)][y])
      +payoff(old[x][y],old[pbc(x-1)][pbc(y-1)])
      +payoff(old[x][y],old[x][pbc(y-1)]);

    /* Calculate the total score. */
    pts[x][y]=pts[x][y]+pt[x][y];

  }
}

/* Output the number of frustrations. */
fprintf(FRUSTRATION,"%d %d\n",i,frustrations/3);

for(i=1;i<=step;i++){

  /* Format the variable. */
  for(j=0;j<=2;j++){
    number[j]=0;
```

```
}
frustrations=0;

for(y=0;y<=_N-1;y++){
  for(x=0;x<=_N-1;x++){

    /* In the case of Natural Strategy */
    if(character[x][y]<=rate_0){
      new[x][y]=natural();
    }

    /* In the case of Perverse Strategy */
    else if(character[x][y]<=rate_0+rate_1){
      new[x][y]=perverse(old[x][y]);
    }

    /* In the case of Spin Strategy */
    else if(character[x][y]<=rate_0+rate_1+rate_2){
      new[x][y]=spin(old[x][y]);
    }

    /* In the case of Backspin Strategy */
    else if(character[x][y]<=rate_0+rate_1+rate_2
            +rate_3){
      new[x][y]=backspin(old[x][y]);
    }

    /* In the case of TitForTat Strategy */
    else if(character[x][y]<=rate_0+rate_1+rate_2
            +rate_3+rate_4){
      new[x][y]=titfortat(old[pbc(x+1)][y],
                          pt[pbc(x+1)][y],
                          old[pbc(x+1)][pbc(y+1)],
                          pt[pbc(x+1)][pbc(y+1)],
                          old[x][pbc(y+1)],
```

```
                        pt[x][pbc(y+1)],
                        old[pbc(x-1)][y],
                        pt[pbc(x-1)][y],
                        old[pbc(x-1)][pbc(y-1)],
                        pt[pbc(x-1)][pbc(y-1)],
                        old[x][pbc(y-1)],
                        pt[x][pbc(y-1)]);
}

/* In the case of Pavlov Strategy */
else if(character[x][y]<=rate_0+rate_1+rate_2
        +rate_3+rate_4+rate_5){
  new[x][y]=pavlov(old[x][y],pt[x][y]);
}

/* In the case of Majority Strategy */
else if(character[x][y]<=rate_0+rate_1+rate_2
        +rate_3+rate_4+rate_5+rate_6){
  new[x][y]=majority(old[pbc(x+1)][y],
                     old[pbc(x+1)][pbc(y+1)],
                     old[x][pbc(y+1)],
                     old[pbc(x-1)][y],
                     old[pbc(x-1)][pbc(y-1)],
                     old[x][pbc(y-1)]);
}

/* In the case of Minority Strategy */
else{
  new[x][y]=minority(old[pbc(x+1)][y],
                     old[pbc(x+1)][pbc(y+1)],
                     old[x][pbc(y+1)],
                     old[pbc(x-1)][y],
                     old[pbc(x-1)][pbc(y-1)],
                     old[x][pbc(y-1)]);
}
```

```
  }
}

/* Prepare PGPLOT. */
for(y=0;y<=_N-1;y++){
  for(x=0;x<=_N-1;x++){
    j=new[x][y];
    plot_x[j][number[j]]=(float)x;
    plot_y[j][number[j]]=(float)y;
    number[j]++;
  }
}

/******** Perform PGPLOT. ********/

/* Open the file. */
sprintf(filename,"step_%04d.gif/GIF",i);
cpgopen(filename);

/* Make the foundation. */
cpgsci(1);
cpgenv(0.0,(float)(_N-1),0.0,(float)(_N-1),0.0,0.0);
cpglab("","","Blue:Stone Red:Scissors Yellow:Paper");

/* Plot Paper. */
cpgsci(7);
cpgpt(number[0],plot_x[0],plot_y[0],18);

/* Plot Scissors. */
cpgsci(2);
cpgpt(number[1],plot_x[1],plot_y[1],18);

/* Plot Stone. */
cpgsci(4);
```

```
cpgpt(number[2],plot_x[2],plot_y[2],18);

/* Close the file. */
cpgclos();

for(y=0;y<=_N-1;y++){
  for(x=0;x<=_N-1;x++){

    /* Metabolize the variable. */
    old[x][y]=new[x][y];

    /* Change the strategy. */
    /* character[x][y]=sprng(); */

    /* Count frustrations. */
    frustrations=frustrations
      +frustration(new[x][y],new[pbc(x+1)][y],
                   new[pbc(x+1)][pbc(y+1)])
      +frustration(new[x][y],new[pbc(x+1)][pbc(y+1)],
                   new[x][pbc(y+1)])
      +frustration(new[x][y],new[x][pbc(y+1)],
                   new[pbc(x-1)][y])
      +frustration(new[x][y],new[pbc(x-1)][y],
                   new[pbc(x-1)][pbc(y-1)])
      +frustration(new[x][y],new[pbc(x-1)][pbc(y-1)],
                   new[x][pbc(y-1)])
      +frustration(new[x][y],new[x][pbc(y-1)],
                   new[pbc(x+1)][y]);

    /* Calculate the score. */
    pt[x][y]=payoff(new[x][y],new[pbc(x+1)][y])
      +payoff(new[x][y],new[pbc(x+1)][pbc(y+1)])
      +payoff(new[x][y],new[x][pbc(y+1)])
      +payoff(new[x][y],new[pbc(x-1)][y])
      +payoff(new[x][y],new[pbc(x-1)][pbc(y-1)])
```

```
          +payoff(new[x][y],new[x][pbc(y-1)]);

        /* Calculate the total score. */
        pts[x][y]=pts[x][y]+pt[x][y];

      }
    }

    /* Output the number of frustrations. */
    fprintf(FRUSTRATION,"%d %d\n",i,frustrations/3);

  }

  for(y=0;y<=_N-1;y++){
    for(x=0;x<=_N-1;x++){

      /* Output the total score. */
      fprintf(POINT,"%d %d %f\n",
              x,y,(float)pts[x][y]/(step+1));

    }
  }

  fclose(FRUSTRATION);
  fclose(POINT);

  return 0;
}



/******** Frustration ********/
int frustration(int first,int second,int third)
{
  int frustration;
```

```
  /* In the case of victory and defeat */
  if((first+1)==(second+1)%3+1&&(first+1)%3+1==(third+1)){
    frustration=1;
  }

  /* In the case of defeat and victory */
  else if((first+1)%3+1==(second+1)&&
          (first+1)==(third+1)%3+1){
    frustration=1;
  }

  /* In the case of others */
  else{
    frustration=0;
  }

  return frustration ;
}


/******** Payoff ********/
int payoff(int i , int you)
{
  int pt;

  /* In the case of victory */
  if((i+1)==((you+1)%3)+1){
    pt=1;
  }

  /* In the case of draw */
  else if(i==you){
    pt=0;
  }
```

```
  /* In the case of defeat */
  else{
    pt=-1;
  }

  return pt;
}


/******** Periodic boundary conditions ********/
int pbc(int room)
{

  /* In the case of left end */
  if(room==-1){
    return room+_N;
  }

  /* In the case of right end */
  else if(room==_N){
    return room-_N;
  }

  /* In the case of others */
  else{
    return room;
  }

}


/******** Natural Strategy ********/
int natural()
{
```

```
  int cell;

  /* Choose at random. */
  cell=(int)(3.0*sprng());

  return cell;
}


/******** Perverse Strategy ********/
int perverse(int cell)
{
  int provisional;

  /* Choose at random. */
  provisional=(int)(3.0*sprng());

  /* Choose at random until it differs from last time. */
  while(cell==provisional){
    provisional=(int)(3.0*sprng());
  }
  cell=provisional;

  return cell;
}


/******** Spin Strategy ********/
int spin(int cell)
{

  /* Choose in order. */
  cell=(cell+2)%3;

  return cell;
```

```
}


/******** Backspin Strategy ********/
int backspin(int cell)
{

  /* Choose in order of reverse. */
  cell=(cell+1)%3;

  return cell;
}


/******** TitForTat Strategy ********/
int titfortat(int mark_0,int pt_0,int mark_1,int pt_1,
              int mark_2,int pt_2,int mark_3,int pt_3,
              int mark_4,int pt_4,int mark_5,int pt_5)
{
  int provisional[6],pt[6],mark[6];
  int cell,i,j=0,k,pt_max=-6;

  mark[0]=mark_0;
  pt[0]=pt_0;
  mark[1]=mark_1;
  pt[1]=pt_1;
  mark[2]=mark_2;
  pt[2]=pt_2;
  mark[3]=mark_3;
  pt[3]=pt_3;
  mark[4]=mark_4;
  pt[4]=pt_4;
  mark[5]=mark_5;
  pt[5]=pt_5;
```

```
  /* Calculate the highest score. */
  for(i=0;i<=5;i++){
    if(pt[i]>=pt_max){
      pt_max=pt[i];
    }
  }

  /* Count the highest score person. */
  for(i=0;i<=5;i++){
    if(pt[i]==pt_max){
      provisional[j]=mark[i];
      j++;
    }
  }

  k=(int)((float)j*sprng());

  /* Choose the highest score person's hand. */
  for(i=0;i<=j;i++){
    if(i==k){
      cell=provisional[i];
    }
  }

  return cell;
}


/******** Pavlov Strategy ********/
int pavlov(int cell,int pt)
{
  int provisional;

  /* If the score is +, choose the same hand. */
  if(pt>0){
```

```
    cell=cell;
  }

  /* If the score is -,
     choose the hand different from last time. */
  else if(pt<0){
    provisional=(int)(3.0*sprng());

    while(cell==provisional){
      provisional=(int)(3.0*sprng());
    }
    cell=provisional;
  }

  /* If the score is 0, choose at random. */
  else{
    cell=(int)(3.0*sprng());
  }

  return cell;
}


/******** Majority Strategy ********/
int majority(int mark_0,int mark_1,int mark_2,int mark_3,
             int mark_4,int mark_5)
{
  int number[3],provisional[3],mark[6];
  int cell,i,j,k=0,l,number_max=0;

  for(j=0;j<=2;j++){
    number[j]=0;
  }
  mark[0]=mark_0;
  mark[1]=mark_1;
```

```
mark[2]=mark_2;
mark[3]=mark_3;
mark[4]=mark_4;
mark[5]=mark_5;

/* Count the number. */
for(j=0;j<=2;j++){
  for(i=0;i<=5;i++){
    if(mark[i]==j){
      number[j]++;
    }
  }
}

/* Calculate the maximum number. */
for(j=0;j<=2;j++){
  if(number[j]>=number_max){
    number_max=number[j];
  }
}

/* Count the group of the maximum number. */
for(j=0;j<=2;j++){
  if(number[j]==number_max){
    provisional[k]=j;
    k++;
  }
}

l=(int)((float)k*sprng());

/* Choose the hand of the maximum number. */
for(j=0;j<=k;j++){
  if(j==l){
    cell=provisional[j];
```

```
    }
  }

  return cell;
}


/******** Minority Strategy ********/
int minority(int mark_0,int mark_1,int mark_2,int mark_3,
             int mark_4,int mark_5)
{
  int number[3],provisional[3],mark[6];
  int cell,i,j,k=0,l,number_min=6;

  for(j=0;j<=2;j++){
    number[j]=0;
  }
  mark[0]=mark_0;
  mark[1]=mark_1;
  mark[2]=mark_2;
  mark[3]=mark_3;
  mark[4]=mark_4;
  mark[5]=mark_5;

  /* Count the number. */
  for(j=0;j<=2;j++){
    for(i=0;i<=5;i++){
      if(mark[i]==j){
        number[j]++;
      }
    }
  }

  /* Calculate the minimum number. */
  for(j=0;j<=2;j++){
```

```
    if(number[j]<=number_min){
      number_min=number[j];
    }
  }

  /* Count the group of the minimum number. */
  for(j=0;j<=2;j++){
    if(number[j]==number_min){
      provisional[k]=j;
      k++;
    }
  }

  l=(int)((float)k*sprng());

  /* Choose the hand of the minimum number. */
  for(j=0;j<=k;j++){
    if(j==l){
      cell=provisional[j];
    }
  }

  return cell;
}
```

A.3 プログラムの実行手順

(i) PGPLOT の準備。

```
setenv PGPLOT_DIR /usr/lib/pgplot
```

(ii) コンパイル。

```
gcc -Wall -c -g delta.c
```

(iii) リンク付け。

```
g77 -o delta.e delta.o -lsprng -lgmp -lcpgplot -lpgplot
  -L/usr/X11R6/lib -lX11 -lgcc -lc
```

(iv) 実行。

```
delta.e
```

(v) GIFMerge の準備。

```
source ~/.tcshrc
```

(vi) GIF ファイルの作成。

```
gifmerge step_*.gif > step.gif
```

謝辞

羽田野先生には、本当にお世話になりました。研究の方向性を見失い欠けたときには、相談に乗っていただき、つまずきを取り除いてくれました。だから、安心して羽田野先生に着いて行けました。とても感謝しています。また、研究室の先輩方のアドバイスや、同級生たちと一緒に考える場があったからこそ、例年より短い期間だったのにも関わらず、研究もここまで進むことができたと思います。ありがとうございます。