卒業研究報告書

題目

二人零和不完全情報ゲームであるジャンケンにおけるゲームの洗練法

指導教員

石水　隆　講師

報告者

12–1–037–0105

長崎　凌

近畿大学理工学部情報学科

平成 28 年 1 月 29 日提出

概要

普段ものごとの決定などをする際に行うジャンケンは二人零和不完全情報ゲームに分類される。二人零和不完全情報ゲームとは、2 人または 2 チームで行い、自分の手を決定する際に相手の手を知ることができないゲームである。(逆に、囲碁や将棋は相手の手を見てから自分の手を決定することができるので完全情報ゲームに分類される。) また、ジャンケンはグー, チョキ, パーの間に優劣は無く、相手が何を出してくるか予想するための材料が無いために、適当に手を出すしか無く、駆け引きや戦略性が全くない。

そこで本研究ではジャンケンに戦略性を生み出すようさまざまな特別ルールを追加することを提案する。また、その中でどの特別ルールが最もゲームを洗練させたかを検証していく。

目次

1　序論

1.1　本研究の背景

現在のジャンケンの基となったものは明治時代頃に日本で成立されたと言われている。そこから今と同じようなグー, チョキ, パーの 3 すくみが用いられていた。3 すくみである理由としては、それぞれひとつの手には優位で、もうひとつの手には劣勢であることで 3 手それぞれが互角でいることができるからである。そうすることで絶対的優位な手が存在しなくなる。また、現在では日本に限らず世界中に広まり、ジャンケンに類似したものが存在している。(例えば、フランスでは手が 3 種類でなく 4 種類のジャンケンも存在する [5])。

1.2　ジャンケンに関する既知の結果

3 人以上でも行うことができるが、本研究では 2 人でのゲームに限定して考える。石 (グー), 鋏 (チョキ), 紙 (パー) の 3 種類の手があり、「ジャンケン」などという掛け声とともに、プレイヤーはおのおの好きな手を両者同時に出す。手はすべて片手で表現でき、石は 5 本すべての指を握り込んだ形であり、鋏は人差し指と中指を伸ばして他の 3 本の指は握った形 1、紙は 5 本すべての指を伸ばして手を開いた形である。石は鋏に勝ち、鋏は紙に勝ち、紙は石に勝つ。両者が異なる手を出せば必ず勝負がつき、同じ手を出した場合は「あいこ」と呼ばれ引き分けである [5]。

ジャンケンのルールは図 1 のように有向グラフを使って表現できる。それぞれの手を頂点で、2 頂点間の勝ち負けを、勝つ頂点から負ける頂点へ向かう有向辺を付与することで表現している。

しかし、ジャンケンは非常に単純なゲームながら対戦相手に勝つための決まった作戦というものは存在していない。ジャンケン以外の 2 人対戦ゲームの代表的なものとして, 将棋を例に挙げる。将棋は、ジャンケンよりはるかに難しいルールや知識が必要であるにもかかわらず、2008 年現在、コンピュータはプロ棋士とも対等に戦えるほどの力を獲得している。これは、2005 年に日本将棋連盟がプロ棋士とコンピュータ将棋との公の場での無断での試合を禁止したことからもうかがうことができる。

こういった将棋に比べジャンケンは、有効な戦略の確保が容易そうであるが、人間相手に戦略などを用いて 5 割以上安定して勝てるというコンピュータはおそらく存在していない。そこには、ジャンケンにおける「勝てる戦略」が明確に存在していない点と、ある程度、運が勝敗を左右してしまうことだと考えられる [6]。

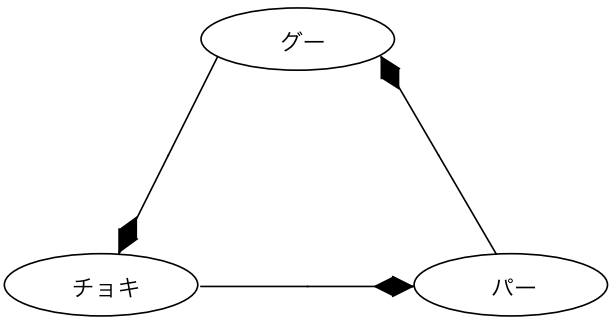

図 1　ジャンケンの有向グラフ表現

1.3 ゲームの洗練度の指標

「ゲームの洗練度」は自由度 B およびゲームのプレイ時間 D を用いて $\sqrt{B}/D$ という値で表される [4]。自由度 B とは、ゲームのある局面においてプレイヤーの出すことができる手の数を表している。ジャンケンの場合、出す手の数はグー, チョキ, パーの 3 種類なので $B = 3$ である。また、プレイ時間 D はゲームの開始から終了までのラウンド数を表している。一般に洗練されたゲームとして評価されている将棋やチェスなどは、洗練度$\sqrt{B}/D$ が 0.07 前後である [4]。洗練度がこの値より大きな値のゲームはチャンスに依存する部分が大きくなり、小さな値のゲームはスキルに依存する部分が大きくなる。したがって、洗練されたゲームであることの必要条件は、洗練度の値が 0.07 に近いことであると言える。

さらに、もうひとつ指標として B^Dという統計量がある。この統計量はゲームの複雑さを示すもので、$\sqrt{B}/D$ がゲームの面白さを表すのに対し、B^Dはゲームの難しさを表す。このふたつの統計量からゲームの洗練度を導くことができる。

また、参考として 2 人で行うゲームで代表的な将棋とテェスの最強レベルのプレイヤーの試合の平均合法手数 (B) と平均終了手数 (D), それを先ほどのふたつの統計量に代入した値を表 1 に示す。

表 1　最強レベルのプレイヤーの試合から得られた数値

	B	D	$\sqrt{B}/D$	B^D
将棋	80	115	0.078	$80^{115} \approx 7.17 \times 10^{218}$
チェス	35	80	0.074	$35^{80} \approx 3.35 \times 10^{125}$

1.4 本研究の目的

先述したように、ジャンケンは昔も現在もゲームとして行われることはあまりなかった。それはジャンケン自体に戦略や駆け引きなどのゲーム性がないからだと考える。ゲーム性が無いのは、ジャンケンに戦略などを生むルールが無いからだと考え、本研究ではさまざまな特別ルールを用意し、それらを組み合わせることでゲーム性を生み出し洗練させる。その中でどのルールの組み合わせが最も洗練させたかを導き、その結果よりゲームを洗練させるにはどのような条件があるのかを理解することを目的とする。

1.5 本報告書の構成

本報告書では、まず 2 節で本研究の内容について説明する。続く 3 節では本研究で得られた結果とそれに対する考察を述べる。そして、最後に 5 節では結論と今後の課題を述べる。

2 研究内容

2.1 研究の方向性

1.3 節で述べたように、洗練されたゲームであるための必要条件として洗練度 $\sqrt{B}/D = 0.07$ と言えるが、難易度 B^Dは明確な数値を述べることができない。何故なら、B^Dはゲームの難易度を示すもので数値が高

ければ高いほど難しいということになる。よって、ゲームの洗練度の指標とはなりにくい。なので、本研究では洗練度が上がると難易度も上がるのか、などの洗練度との相関関係を見るため用いる。

本研究では、ジャンケンの洗練度を上記の値に近づけるため、種々の特別ルールを付加する。

2.2 特別ルール

本研究では、ジャンケンの特別ルールとして、出す手によって得点が変わるルールを 4 パターンと、ゲームの終了条件ルールを 6 パターン提案する。そして、提案する特別ルールの詳細を以下に述べる。

2.2.1 出す手によって得点が変わるルール

1 つ目に提案するルールは、出す手によって得点が変化するルールである。普段のジャンケンとは違い、それぞれ出す手によって得点が違うので相手は高得点の手を出してくるかや、その裏をかくかなどの駆け引きを行うことができる。

本研究では以下に示す 4 パターンを提案する。

- H1. グー:5 , チョキ:3 , パー:1
- H2. グー:5 , チョキ 1: , パー:3
- H3. グー:10 , チョキ:1 , パー:1
- H4. グー:10 , チョキ:10 , パー:1

各得点パターンの特徴を以下に挙げる。

- H1: 最も得点が高いグーに勝つパーが最も得点が低い。よって、グーを出すリスクが少ない。
- H2: H1 と同様でグーが最も高い。しかし、グーに勝つパーが今回は 3 点で、リスクが高い。
- H3: グーが圧倒的高得点で他のチョキとパーが同じ 1 点。相手がグーを出すと読んでパーで地道に稼ぐのか、その裏をかいてグーで高得点を狙うのか。
- H4: H3 に対してグーとチョキが同じ 10 点で高得点。そしてパーが 1 点で低い。なのでパーを出すメリットがあまり無い。そして同じ 10 点であるグーとチョキで優位なグーが最もリスクの少ない手。

2.2.2 終了条件ルール

2 つ目に提案するのは、ゲームの終了条件を変えるルールである。このルールには大きく分けて 3 種類あり、ラウンドマッチルールと○点先取ルール、点差コールドルールである。ラウンドマッチルールは、指定のラウンド数だけジャンケンを行い、終了した時点で点数が高い方が勝利というルールである。○点先取ルールは、例えば 10 点先取の場合、先に 10 点かそれを超える点数を取った瞬間に勝利し、ゲームが終了するというルールである。最後に点差コールドルールは、例えば 5 点差コールドの場合、両者の点差が 5 点かそれ以上になった時点で点数が高い方が勝利し、ゲームが終了するというルールである。それぞれ 3 種類のルールの詳細を以下に述べる。

- E1. 5 ラウンド勝負
- E2. 10 ラウンド勝負

- E3. 10 点先取
- E4. 20 点先取
- E5. 5 点差コールド
- E6. 10 点差コールド

各終了条件の特徴を以下に挙げる。

- E1: 短期決戦なのでお互い大きい得点を狙うはず。その裏をかくか、その裏の裏で大きい得点を狙うか。
- E2: E1 に比べてラウンド数が多いので余裕がある。なので数回大きい得点を取って逃げ切るか、小さい得点を地道に取り続けて勝つか。
- E3: 得点パターンによっては 1 回か 2 回で終わってしまう場合がある。なのでそれを阻止する手を出しつつ、どこで自分が攻めていくか。
- E4: E3 に比べて即決着がつくことはない。なので相手の大きい得点を防ぎつつ時々自分が大きい得点を狙うのか、小さい得点でも高確率で勝ってくのか。
- E5: これも 1 回で決着がついてしまうことがある。なので、相手の一発勝利をとりあえず阻止するのか、裏をかいてそのまま自分が大きい得点を狙いにいくのか。
- E6: これも得点パターンによっては 1,2 回で決着がつく。なのでとりあえず相手の一発勝利を阻止し、どこかで攻めるのか、小さい得点でコツコツ点差を開けていくのか。

2.3 計算機実験

本研究では、2.2 節で述べた特別ルールの洗練度および難易度を計算するために計算機実験を行った。計算機実験で用いたプログラムソースは付録に示す。

また、本研究では 2.2 節で述べた特別ルールの全ての組み合わせ 24 パターンに対して、コンピュータ同士の対戦を各 1000 回ずつ行った。

2.4 計算機の用いた戦略

本研究の計算機で用いた戦略は、”各手に付加された得点に比例およびそれが負ける手に付加された得点に反比例して出す”というものである。詳しく述べると、例えば、グー:5, チョキ:3, パー:1 という得点パターンだとする。するとグーが 5 点で高得点である。なのでグーを出したくなるが相手もそれを読んでパーを出してくる可能性がある。そういう場合に、グーを負かす手の得点を見る。今回はパーの得点を見て、1 点なので負けてもリスクが少ない。なのでグーを出す価値はある。よって、この戦略は負けた際のリスクで出す手を考える戦略である。

3 実験結果および考察

表 2 に各ルールでの平均終了ラウンド数、表 3 には $\sqrt{B}/D$ の値、表 4 には B^Dの値を示す (小数第 3 位で四捨五入)。表 3 より、洗練度$\sqrt{B}/D$ を指標とした場合、$(H1:E4), (H1:E6), (H2:E4)$ の組み合わせが 0.06 で最も 0.07 に近い。よって、ゲームの終了条件としては 20 点先取および 10 点差コールドとすると良

いと考えられる。また、ラウンドマッチルールである $E1, E2$ の値を見てみると 0.07 を大きく上回っていることが分かる。これはチャンスに依存している、つまり運に大きく左右されるということである。よって、ラウンドマッチルールはあまり洗練法には向かないことがわかる。

そして、得点ルールの違いによる洗練度の差はあまり見られないことから、ゲームの洗練度は得点よりも終了条件に依存していると言える。

一方、難易度 B^D に関しては、チェスや将棋の値に比べて遥かに低く、ラウンドマッチジャンケンは極めて単純なゲームであることが示される。

ラウンドマッチジャンケンのような同じことを繰り返す単純なゲームは難易度が低くなる。また、チェスや将棋のような手が進むにつれて盤面が変化し、各手番でいろいろな手を考えなくてはならない複雑なゲームは難易度が高くなる。それを洗練度の視点から見てみると、チェスや将棋はかなり洗練度の高いゲームだと言えるが、ラウンドマッチジャンケンはあまり高いとは言えない。よって、本研究で洗練度の基準にしていた 0.07 という数値はジャンケンのような難易度の低いゲームには当てはめることができないと推測できる。

表 2　各ルールでの平均終了ラウンド数

	E1	E2	E3	E4	E5	E6
H1	3.4	6.9	12.7	26.7	10.3	28.4
H2	3.3	7.0	13.6	26.0	12.4	34.7
H3	4.3	7.6	16.3	35.9	17.5	34.1
H4	3.6	5.1	10.3	18.9	6.4	14.6

表 3　各ルールでの洗練度

	E1	E2	E3	E4	E5	E6
H1	0.50	0.25	0.13	0.06	0.16	0.06
H2	0.52	0.24	0.12	0.06	0.13	0.04
H3	0.40	0.22	0.10	0.04	0.09	0.05
H4	0.48	0.33	0.16	0.09	0.27	0.11

表 4　各ルールでのゲームの難易度

	E1	E2	E3	E4	E5	E6
H1	4.19×10	1.95×10^3	1.14×10^6	5.48×10^{12}	8.21×10^4	3.55×10^{13}
H2	3.75×10	2.18×10^3	3.08×10^6	2.54×10^{12}	8.24×10^5	3.59×10^{16}
H3	1.12×10^2	4.22×10^3	5.98×10^7	1.34×10^{17}	2.23×10^8	1.86×10^{16}
H4	5.22×10	2.71×10^2	8.21×10^4	1.04×10^9	1.13×10^3	9.24×10^6

4　結論・今後の課題

4.1　結論

本研究ではジャンケンの特別ルールの追加による洗練を行った。洗練度の観点から見ると、20 点先取および 10 点差コールドルールを導入することでジャンケンがより面白くなると考えられる。また、先述したようにゲームの洗練度は得点より終了条件に依存していることから、より多く細かい終了条件を用意することで、さらに洗練度を高めることが期待できる。

4.2　今後の課題

$\sqrt{3}/D = 0.07$ として方程式を解くと $D = 24$ となる。よって、終了ラウンド数を 24 前後に固定するルールを追加することでゲームをより洗練化できることになる。しかし、それは考えにくい。なぜなら、ゲーム性に関係なく 24 ラウンドでゲームを終了させれば最も洗練されたことになるからである。

よって今後の課題は、$\sqrt{B}/D$ に代わる洗練度の指標を他に見つけ出し、それらの数値を加味して改めて最も洗練化させる特別ルールを導きだすことが挙げられる。そして、そのジャンケンがはたして他のジャンケンより面白いのか実際に人にしてもらって検証することも必要である。そして、ジャンケンのような難易度の低い単純なゲームの洗練度を測る $\sqrt{3}/D$ の値を求めることである。

謝辞

本報告書を作成するにあたり、数えきれない程のご指導、ご助力などたいへん尽力していただき、石水隆講師には心から感謝申し上げます。本当にお世話になりました。

参考文献

1) 飯田　弘之 ： ゲーム研究のいま, 情報の科学と技術　62 巻　12 号 ,pp.527-532 (2012),
https://dspace.jaist.ac.jp/dspace/bitstream/10119/10889/1/19097.pdf
2) 漆間　幸雄, 飯田　弘之 ： ゲーム洗練度の理論とポーカー , ゲームプログラミングワークショップ 2005 論文集 ,2005(15) ,pp.138-141 (2005),
http://id.nii.ac.jp/1001/00097591/
3) 柏木　理志, 飯田　弘之: ゲームの洗練法 -ジャンケンを題材として- , 情報処理学会研究報告 2003-GI-010 ,pp.9-13 (2003),
http://id.nii.ac.jp/1001/00058569/
4) H.Iida, N.Takeshima and J.Yoshimura: A Metric for Entertainment of Boardgames: its implication for evolution of chess variants, IWEC 2002 Proceedings ,pp.65-72 (2003)
5) 伊藤大雄: 一般化ジャンケン , オペレーションズ・リサーチ ,Vol.18 ,No.3, pp.156-160 (2013),
2013,http://www.orsj.or.jp/archive2/or58-03/or58_3_156.pdf
6) 山下将臣: じゃんけんゲームに対する遺伝的アプローチ , 高知工科大学 情報システム工学科 平成 20 年度学士学位論文 (2009),
http://www.kochi-tech.ac.jp/library/ron/2008/2008info/1090396.pdf
7) 服部太輔, 細江政範, 石黒友一: ジャンケンの文化的側面と数理解析 , 南山大学 数理情報学部 数理科学科 2006 年度卒業研究 (2007),
http://www.kochi-tech.ac.jp/library/ron/2008/2008info/1090396.pdf
8) 牧野泰裕, 西野順二, 小高知宏, 小倉久和: 繰り返し対戦型じゃんけんにおける戦略の特徴 , 福井大学 工学部 研究報告 第 45 巻 第 1 号 ,pp.71-79 (1997),
http://repo.flib.u-fukui.ac.jp/dspace/bitstream/10098/3425/1/AN00215401-045-01-007. pdf
9) 佐々木宣介, 橋本剛, 梶原羊一郎, 飯田弘之: チェスライクゲームにおける普遍的指標, 情報処理学会　研究報告ゲーム情報学 ,Vol.1999-GI-001 ,No.53, pp.91-98 (1999),
http://id.nii.ac.jp/1001/00058670/

付録 A　ソースプログラム

以下に本研究で作成したコンピュータ同士がジャンケンを行う Roshambo クラスを示す。

```
package sotuken;

import java.util.Scanner;
import java.util.Random;

/**
 * ジャンケンゲーム
 */
public class Roshambo {
        static final int GU = 0;     // グー
    static final int CH = 1;     // チョキ
    static final int PA = 2;     // パー
    static final int WIN = 1;    // 勝ち
    static final int LOSE = -1; // 負け
    static final int DROW = 0;  // 引き分け
    static final int[][] isWin= {{DROW, WIN,  LOSE}, // 勝敗判定
                                 {LOSE, DROW, WIN},
                                 {WIN,  LOSE, DROW}};
    static final String[] handToString = グー{"", チョキ"", パー""}; // 手の文字列表現

    int round;   // 対戦ラウンド数
    int[] score; // 得点
    int[] victries; // 勝利数
    int[] svictries; // 連勝数
    int maxRound;    // 最大ラウンド数
    int boundScore;  // 勝利となる規定得点
    int limitDiffScore; // 勝利となる得点差
    int[] handBonus; // 出した手によるボーナス
    static int difRound = 0;

    static final int handBonusType = 4;  // 出した手によるボーナスのタイプ
    /*グー
        0:1 チョキ1 パー1グー
        1:5 チョキ3 パー1グー
        2:5 チョキ1 パー3グー
        3:10 チョキ1 パー1グー
        4:10 チョキ10 パー1
     */
    static final int comboBonusType = 0; // 連勝数によるボーナスタイプ
    /*連勝ボーナス無し
        0:連勝以上で
        1:2+1連勝以上で
        2:3+3連勝で
        3:2+1, 連勝で3+2, 連勝で4+3, 連勝で5+4, ...連勝で
        4:2+1, 連勝で3+2, 連勝で4+4, 連勝で5+8, ...連勝で
        5:2*2, 連勝で3*3, 連勝で4*4, 連勝で5*5, ...連勝で
        6:2*2, 連勝で3*4, 連勝で4*8, 連勝で5*16, ...
```

```
*/
static final int gameSetType = 1;     // 終了条件のタイプ
/*回勝負
    0:1回勝負
    1:5回勝負
    2:10点先取
    3:10点先取
    4:20点差コールド
    5:5点差コールド
    6:10
 */

static final int comLevel = 2; // の戦略レベルCOM
/*ランダム
    0:各手に付加された得点に比例して出す
    1:各手に付加された得点に比例およびそれが勝つ手に付加された得点に比例して出す
    2:各手に付加された得点に比例およびそれが負ける手に付加された得点に反比例して出す
    3:それ以外常にグー
    :
*/

int[] weight; // 各手を出す確率に付加する重み

Scanner keyBoardScanner; // キーボードスキャナ
Random rnd; // 乱数発生用

/**
 * コンストラクタ
 * 各初期値をセットする
 */
Roshambo() {
    round = 0;            // ラウンド数
    score = new int[2];   // 得点
    score[0] = 0;
    score[1] = 0;
    victries = new int[2]; // 勝利数
    victries[0] = 0;

    victries[1] = 0;
    svictries = new int[2]; // 連勝数
    svictries[0] = 0;
    svictries[1] = 0;
    handBonus = new int[3];
    switch (handBonusType) { // 出した手によるボーナス
    case 0: // グー1 チョキ1 パー1
        handBonus[GU] = 1;
        handBonus[CH] = 1;
        handBonus[PA] = 1;
        break;
    case 1: // グー5 チョキ3 パー1
        handBonus[GU] = 5;
        handBonus[CH] = 3;
        handBonus[PA] = 1;
```

```
        break;
    case 2: // グー5 チョキ1 パー3
        handBonus[GU] = 5;
        handBonus[CH] = 1;
        handBonus[PA] = 3;
        break;
    case 3: // グー10 チョキ1 パー1
        handBonus[GU] = 10;
        handBonus[CH] = 1;
        handBonus[PA] = 1;
        break;
    case 4: // グー10 チョキ10 パー1
        handBonus[GU] = 10;
        handBonus[CH] = 10;
        handBonus[PA] = 1;
        break;
    default: // グー1 チョキ1 パー1
        handBonus[GU] = 1;
        handBonus[CH] = 1;
        handBonus[PA] = 1;
        break;
    }
    switch (gameSetType) { // ゲーム終了条件
    case 0: // 回勝負1
        maxRound = 1;
        boundScore = Integer.MAX_VALUE;
        limitDiffScore = Integer.MAX_VALUE;
        break;
    case 1: // 回勝負5
        maxRound = 5;
        boundScore = Integer.MAX_VALUE;
        limitDiffScore = Integer.MAX_VALUE;
        break;
    case 2: // 回勝負10
        maxRound = 10;
        boundScore = Integer.MAX_VALUE;
        limitDiffScore = Integer.MAX_VALUE;
        break;
    case 3: // 点先取10
        maxRound = Integer.MAX_VALUE;
        boundScore = 10;
        limitDiffScore = Integer.MAX_VALUE;
        break;
    case 4: // 点先取20
        maxRound = Integer.MAX_VALUE;
        boundScore = 20;
        limitDiffScore = Integer.MAX_VALUE;
        break;
    case 5: // 点差コールド5
        maxRound = Integer.MAX_VALUE;
        boundScore = Integer.MAX_VALUE;
        limitDiffScore = 5;
        break;
```

```
        case 6: // 点差コールド10
            maxRound = Integer.MAX_VALUE;
            boundScore = Integer.MAX_VALUE;
            limitDiffScore = 10;
            break;
        default : // 回勝負1
            maxRound = 1;
            boundScore = Integer.MAX_VALUE;
            break;
        }
        weight = new int[3];
        setWeight (comLevel); // 各手を出す重みを設定する

        keyBoardScanner = new Scanner (System.in); // 入力をキーボードに設定
        long seed = System.currentTimeMillis(); // 現在時刻から乱数の種を生成
        rnd = new Random (seed); // 乱数発生用
    }

    /**
     * メインメソッド
     */
    public static void main (String[] args) {
        for(int i = 0; i < 10000; i++) {
        Roshambo roshambo = new Roshambo(); // インスタンス生成
        roshambo.play(); // ゲーム開始
        }
        /*difRound = difRound ;*/
        System.out.println平均("" + difRound + ラウンド"");
    }

    /**
     * ゲームを開始する
     */
    void play() {
        int[] hand = new int[2];
        while (round < maxRound  // 最大ラウンド数以内
            && score[0] <= boundScore && score[1] <= boundScore  // 規定得点以下
            && (score[0] - score[1]) <= limitDiffScore
            && (score[1] - score[0]) <= limitDiffScore) { // 規定得点差以下
            //showResult(); // 現在の結果表示
            hand[0] = com(comLevel); // プレイヤーの手入力
            hand[1] = com (comLevel); // の手選択COM
            /*System.out.println あなた(" : " + handToString[hand[0]]
             * + " 私    : " + handToString[hand[1]]); */
            int result = isWin[hand[0]][hand[1]]; // 勝敗判定
            int winner = -1; // 勝者
            if (result == WIN) {
                winner = 0;
                //System.out.println あなたの勝ちです("");
            } else if (result == LOSE) {
                winner = 1;
                //System.out.println 私の勝ちです("");
            } else {
```

```
            //System.out.println 引き分けです("");
        }
        if (result == WIN || result == LOSE) {
            ++victries[winner];       // 勝利数増加
            ++svictries[winner];      // 連勝数増加
            svictries[1-winner] = 0; // 連勝数リセット
            int handScore = handBonus[hand[winner]];
            int comboScore = 0;
            switch (comboBonusType) { // 連勝ボーナスに応じた得点追加
            case 0: // 連勝ボーナス無し
                /*System.out.println 得点(" : +" + handScore); */
                score[winner] += handScore;
                break;
            case 1: // 連勝以上で2+1
                if (svictries[winner] >= 2) {
                    System.out.println 得点(" : +" + handScore + " 連勝ボーナス  : +1");
                    score[winner] += (handScore + 1);
                } else {
                    System.out.println 得点(" : +" + handScore);
                    score[winner] += handScore;
                }
                break;
            case 2: // 連勝以上で3+3
                if (svictries[winner] >= 3) {
                    System.out.println 得点(" : +" + handScore + " 連勝ボーナス  : +3");
                    score[winner] += (handScore + 3);
                } else {
                    System.out.println 得点(" : +" + handScore);
                    score[winner] += handScore;
                }
                break;
            case 3: // 連勝で2+1, 連勝で3+2, 連勝で4+3, 連勝で5+4, ...
                if (svictries[winner] >= 2) {
                    comboScore = svictries[winner]-1;
                    System.out.println 得 点(" : +" + handScore + " 連 勝 ボ ー ナ ス
: +" + comboScore);
                    score[winner] += (handScore + comboScore);
                } else {
                    System.out.println 得点(" : +" + handScore);
                    score[winner] += handScore;
                }
                break;
            case 4: // 連勝で2+1, 連勝で3+2, 連勝で4+4, 連勝で5+8, ...
                if (svictries[winner] >= 2) {
                    comboScore = 1 << (svictries[winner]-1);
                    System.out.println 得 点(" : +" + handScore + " 連 勝 ボ ー ナ ス
: +" + comboScore);
                    score[winner] += (handScore + comboScore);
                } else {
                    System.out.println 得点(" : +" + handScore);
                    score[winner] += handScore;
                }
                break;
```

```
                case 5: // 連勝で2*2, 連勝で3*3, 連勝で4*4, 連勝で5*5, ...
                    if (svictries[winner] >= 2) {
                        comboScore = svictries[winner];
                        System.out.println 得 点(" : +" + handScore + " 連 勝 ボ ー ナ ス
: *" + comboScore);
                        score[winner] += (handScore * comboScore);
                    } else {
                        System.out.println 得点(" : +" + handScore);
                        score[winner] += handScore;
                    }
                    break;
                case 6: // 連勝で2*2, 連勝で3*4, 連勝で4*8, 連勝で5*16, ...
                    if (svictries[winner] >= 2) {
                        comboScore = 1 << (svictries[winner]-1);
                        System.out.println 得 点(" : +" + handScore + " 連 勝 ボ ー ナ ス
: *" + comboScore);
                        score[winner] += (handScore * comboScore);
                    } else {
                        System.out.println 得点(" : +" + handScore);
                        score[winner] += handScore;
                    }
                    break;

                default: // 連勝ボーナス無し
                    System.out.println 得点(" : +" + handScore);
                    score[winner] += handScore;
                    break;
                }
            } else { // 引き分けの場合
                svictries[0] = 0; // 連勝数リセット
                svictries[1] = 0;
            }
            ++round;
            if(maxRound - round < score[0] - score[1]
                        || maxRound - round < score[1] - score[0]) {
                break;
            }
        }
        difRound += round;

        /*showResult(); // 結果表示
        if (score[0] > score[1]) System.out.println あなたの勝ちです("");
        else if (score[0] < score[1]) System.out.println 私の勝ちです("");
        else System.out.println 引き分けです(""); */
    }

    /**
     * 結果表示
     */
    void showResult() {
        if (round == 0) {
            System.out.println 試合開始("");
        } else {
```

```
        System.out.println (round + 回戦終了"");
    }
    System.out.print (score[0] + 対"" + score[1] + "   ");
    System.out.print (victries[0] + 勝"" + victries[1] + 敗"   ");
    if (svictries[0] > 0) System.out.println (svictries[0] + 連勝中"");
    else if (svictries[1] > 0) System.out.println (svictries[1] + 連敗中"");
    else System.out.println();
}

/**
 * キーボードからプレイヤーの手を入力する
 * @return int : プレイヤーの入力した手
 */
int player () {
    int hand = -1;
    while (true) { // 適切な値が選択されるまでループ
        System.out.print 手を入力してください(" グー(0: チョキ1: パー2:) : ");
        String inputString = keyBoardScanner.next();
        try {
            hand = Integer.parseInt (inputString);
        } catch (NumberFormatException e) { // 整数値以外が入力された場合
            System.out.println のいずれかを入力してください("0,1,2");
            continue;
        }
        if (hand <0 || 2<hand) {
            System.out.println のいずれかを入力してください("0,1,2");
            continue;
        }
        break;
    }
    return hand;
}

/**
 * が手を選択するCOM
 * @param int level : のレベルCOM
 * @return int : が選択した手COM
 */
int com (int level) {
    int hand = -1;
    switch (level) { // のレベルに応じた手を選択COM
    case 0: // ランダムに選択
        hand = rnd.nextInt (3);
        break;
    case 1: // 各手に付加された得点に比例して出す
    case 2: // 各手に付加された得点に比例およびそれが勝つ手に付加された得点に比例して出す
    case 3: // 各手に付加された得点に比例およびそれが負ける手に付加された得点に反比例して出す
        int r = rnd.nextInt (weight[PA]);
        if (r < weight[GU]) hand = GU;
        else if (r < weight[CH]) hand = CH;
        else hand = PA;
        break;
    case 4:
```

```
            /* レベル毎にの戦略に応じた値を選択するプログラムを以下ここに加えるCOM */
        default: // 常にグーを選択戦略の良し悪しの比較用()
            hand = GU;
            break;
        }
        return hand;
    }

    /**
     * が各手を選択する重みを設定するCOM
     * @param int level : のレベルCOM
     */
    void setWeight (int level) {
        switch (level) {
        case 0: // 各手を均等に出す
            weight[GU] = 1;
            weight[CH] = 2;
            weight[PA] = 3;
            break;
        case 1: // 各手に付加された得点に比例して出す
            weight[GU] = handBonus[GU];
            weight[CH] = weight[GU] + handBonus[CH];
            weight[PA] = weight[CH] + handBonus[PA];
            break;
        case 2: // 各手に付加された得点に比例およびそれが勝つ手に付加された得点に比例して出す
            weight[GU] = handBonus[GU] * handBonus[CH];
            weight[CH] = weight[GU] + handBonus[CH] * handBonus[PA];
            weight[PA] = weight[CH] + handBonus[PA] * handBonus[GU];
            break;
        case 3: // 各手に付加された得点に比例およびそれが負ける手に付加された得点に反比例して出す
            weight[GU] = handBonus[GU] * handBonus[GU] * handBonus[CH];
            weight[CH] = weight[GU] + handBonus[CH] * handBonus[CH] * handBonus[PA];
            weight[PA] = weight[CH] + handBonus[PA] * handBonus[PA] * handBonus[GU];
            break;
        case 4:
            /* レベル毎にの戦略に応じた重みの設定を以下ここに加えるCOM */
        default: // グーのみ出す
            weight[0] = 1;
            weight[1] = 1;
            weight[2] = 1;
            break;
        }
    }
}
```