卒業研究報告書

題目

確率依存型ゲームにおける洗練度

指導教員

石水　隆　講師

報告者

11-1-037-0007

金子　健吾

近畿大学理工学部情報学科

平成 28 年 1 月 29 日提出

概要

世界には多くのゲームが存在する。将棋やチェスなどのゲームは世界中で人気を博し、長年親しまれ続けている。それらのゲームの面白さの理由はなんなのか。どのようなゲームが一番面白いと言えるのだろうか。また、その面白さを測る方法はないのか。

例えば、ゲームの難易度を推量するためのある方法が存在する。手法としてはゲームの平均可能手数 B、平均終了手数 D に対し B^D で近似するというもので、求められる値が小さければ、容易に先読みができ勝者を予想する事ができるというわけである。つまり、ゲームとしての魅力に欠け、ユーザーに飽きられやすいゲームだということがわかるのである。この式を用いれば、「退屈なゲーム」の排斥は可能である。一方、B^D の値が大きい場合、複雑で難易度の高いゲームになる。しかし、この方法を用い淘汰されたゲームを実際に遊んだ時に必ずしも面白くないと言えるであろうか？なによりも重要なのは実際に人間が遊んだ時に面白いと感じられるか否かであるのではないだろうか。

一方、ゲームの洗練度という考え方がある。ゲームの洗練度とはゲームとしての面白さを推定するための方法である。指標として、洗練度$\sqrt{B}/D$が用いられる 1)。ここで B は各局面での平均自由度、D はゲーム終了までの手数。チェスや将棋、麻雀などの洗練度$\sqrt{B}/D$は 0.07 前後を示しており、この値がゲーム性の高いバランスのとれたゲームである可能性を示唆している 1)。

本論文ではゲームの洗練度$\sqrt{B}/D$という考え方を用い、実際の面白さとの間に相関があるのか。また、隔たりがあればその原因を検討する。

目次

1　序論

1.1 ゲームの洗練度

ゲームの難易度推定法としてそのゲーム木の大きさを指標として用いる場合がある。ゲーム木の大きさは、そのゲームの平均終了手数 D と平均可能手数 B に対し B^D で近似される[6]。コンピュータプログラムを作成するために、先読み探索アルゴリズムが重要となるゲーム研究においては、B^D はそのゲームの難しさを特徴づける重要な要素である[8]。この B^D の値が小さければ、ゲームの初期状況から最終局面までの先読みを行い、解く事ができる簡単なゲームということになる。逆に B^D の値大きいほど複雑で難易度の高いゲームになるというわけである。この難易度推定式 B^D は数多くのゲームが誕生する中で、そのゲームが生き残れるかどうかの 1 つの基準になっている 6)。しかし、この B^D だけではゲームの面白さを測るには不十分である。この式 B^D で求めた値が大きいにもかかわらず、面白くないゲームが存在するのも自明のことである。そうなると否が応でも出てくる疑問は、ゲームの面白さとは何かという疑問である。

一方、難易度推定式 B^D 以外のゲームのとしての面白さを推定する指標として、$\sqrt{B}/D$ で定義される洗練度がある[1]。洗練度推定式 $\sqrt{B}/D$ はチャンスとスキルのバランスが共存する洗練されたゲームを表す指標であるとされている[1]。チェスや将棋の洗練度の値は一様に 0.07 付近であり、この値が洗練されたゲームの一種の基準と考えられる[3]。ここでいう「チャンス」を持つゲームとは、圧倒的不利な状態から態勢を持ち直す可能性を持ち合わせていること。また「スキル」を持つゲームとは、自分の技術、戦略性をもってして態勢を変えられる要素を持つゲームを指す。その 2 つの要素が並存してこそ、面白いゲームといえる[6]。

本研究では、ゲームの洗練度と実際の面白さとの間に相関があるのか。また、隔たりがあればその原因がなんであるのか検討する。

1.2 ラウンドマッチジャンケン

本研究では洗練度と実際の面白さの相関を検証するために、題材としてラウンドマッチジャンケンを用いた。一般的にジャンケンは、一回勝負の確率依存型ゲームである。各手(グー、チョキ、パー)を出す確率はそれぞれ $1/3$ と均一で、運に依存する。一方、題材に選んだラウンドマッチジャンケンは、複数回のゲームによる総合結果から勝利を決めるというルールを追加したジャンケンである。ラウンドマッチ制にする事で運任せのゲームから脱し、何かしらの面白さが加わるのではないかと考えたため、このラウンドマッチジャンケンを洗練度の検証に用いる。

1.3 ジャンケンに関する既知の結果

ここでジャンケンについての既知の結果を示す。ジャンケンは明治時代中期、九州を中心とした西日本に多く残っている拳遊びに、数拳の三すくみの拳の要素が新たに加えられ考案されたと考えられている[10]。ジャンケンの基本的ルールは、おそらくほとんどの日本人にとってなじみ深いものであり、ルールを知らない人は少ないと思われる。グー、チョキ、パーの三種類の手があり、「じゃんけんぽん」などの合図に合わせ、各々が好きな手を出す[8]。グーであればチョキに、チョキであればパーに、パーであればグーに勝つことができる。この関係は図 1 のような有向グラフに表すことができる[10]。

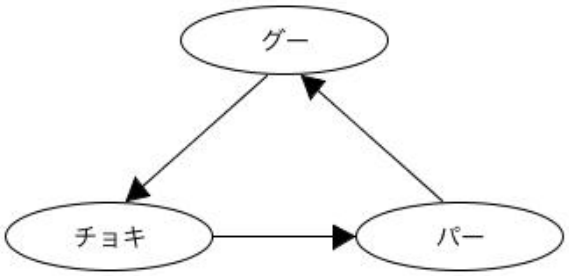

図 1:有向グラフによる手の関係

本研究では、コンピュータ同士の対戦と人間同士の対戦を実施する。しかし、コンピュータと人間の手の出し方には遺伝的観点から見た場合に違いが現れることがわかっている[9]。人間が手を出すときには、ある種の傾向が出る。例えば、あるプレイヤーに何度かジャンケンをさせた場合、明らかにチョキを多く出すなど個人の癖が出てくるのである。また、対戦相手はその手の出し方を記憶し、それに勝てる手を出すような傾向が出る可能性もある。これらは無意識的、意識的の両方の場合がある。繰り返し対戦型では、コンピュータの手の出し方をそれらの人間特有の癖に則ってプログラムすることで、戦略性や癖のある手の出し方をさせることも可能ではある。今回用いるラウンドマッチジャンケン、つまり繰り返し対戦型ジャンケンというルールについては履歴学習型戦略という戦略を用い、コンピュータが戦略的にゲームをできることがわかっている[11]。本研究では、各手を出す確率を同じく 1/3 と仮定し前後の勝負とは切り離された確率依存型ゲームとして扱う。

1.4 調査方法

本研究では$\sqrt{B}/D$によって表される洗練度と「実際の面白さ」との関係を検証する。そこで「実際の面白さ」を測定するために、作成したゲームを実際に被験者が遊んでもらい、面白さを比較してもらう。また、被験者からゲームで感じた不満点や改善点を聞き取り、その結果を反映させてより面白くなるようにゲームのルールを改変していく。

1.5 本研究の目的

本研究の目的はゲームの洗練度$\sqrt{B}/D$と実際に遊んだ時に感じる面白さの間に相関があるのかを検証することである。また、難易度 B^D と実際の面白さの間に相関があるのか検証することも目的である。

1.6 報告書の構成

本報告書の構成は以下の通りである。まず、第 2 章では実験の内容と検証過程を述べる。そして、第 3 章では実験結果をさらに詳しく報告、考察する。第 4 章では本研究で得た結果からどのような課題が残されているかを述べる。

2　研究内容

本章では、洗練度と実際の面白さの間にある相関の検証過程を詳しく述べる。

2.1 ラウンドマッチジャンケン

1.2 節で述べたように、ラウンドマッチジャンケンは、ジャンケンに複数回のラウンドで勝敗を決めるというルールを追加したものである。

洗練度と実際の面白さの相関を検証するにあたり、本研究では二種類のラウンドマッチジャンケンを用意した。ラウンドマッチジャンケンのゲームとしての基本的な流れは以下の通りである。

- 各ラウンドで 1 回ジャンケンを行う。ジャンケンに勝利したプレイヤーは勝ち点を 1 ポイント獲得。負けたプレイヤーは点を得られない。また、引き分けの場合は両プレイヤーともに勝ち点を獲得できない。規定のゲーム終了条件を満たした時点で、ゲーム終了。勝ち点の大きいプレイヤーが勝者となる。

本研究では、ゲーム終了条件に関するルールとして、以下に示す R1、R2 の 2 つのルールを用いる。

- R1：「n 回先に勝った方が勝ち」
- R2：「n 回中多くの回数勝った方が勝ち」

すなわち、ルール R1 は n 点先取制であり、ルール R2 は n ラウンドマッチである。
このように R1、R2 という二つのゲームを作成した理由は「洗練度が 0.07 に近いもの同士を比較した場合に実際の面白さに違いは出るか」、「洗練度が 0.07 に近いものと離れているもの二つを比較し実際の面白さに違いは出るのか」、という 2 つの観点から洗練度と実際の面白さの相関を検証するためである。研究初期段階では、1 種類のルールの中で検証しようと試みたがルールとしての差が小さく、実際に遊んだ時の評価に差がでない結果となったため、2 種類のゲームを用いて検証することにした。

2.2 ラウンドマッチジャンケンプログラム

本研究では、ラウンドマッチジャンケンの洗練度について検証するために、Java を用いてラウンドマッチジャンケンプログラムを作成した。付録 1 に本研究で作成したプログラムのソースを示す。

2.2.1　各クラスの役割

今回作成したプログラムは以下の 5 つのクラスから成る。

- Janken クラス:ジャンケン実行クラス
- Judge クラス:勝負を判定する審判クラス
- Player クラス:ジャンケンをするプレイヤークラス
- RandomTactics クラス: Tactics クラスを継承した、ランダムに手を出す戦略クラス
- Tactics クラス:ジャンケンの戦略インターフェース

2.2.2　Judge クラスの各メソッド

Judge クラスはジャンケンの実行、判定を行うクラスである。Judge クラスにはジャンケンの勝敗を判定する幾つかのメソッドがある。表 1 に各メソッドの役割を示す。

表 1 Judge クラスの主なメソッド

メソッド	処理内容
void startJanken(int round, int doCount, int combo, Player player1, Player player2)	ジャンケンを実行するメソッド
Player judgeJanken(Player player1, Player player2)	一回ごとのジャンケンの勝敗を判定するメソッド
Player judgeFinalWinner(Player player1, Player player2)	ゲームの最終勝者を判定メソッド

judgeJanken(Player player1, Player player2)は、一回ごとのジャンケンを判定するメソッドであ

る。引数として Player インスタンスの player1、player2 を受け取り、プレイヤーの showhand()メソッドを用い、それぞれの手を見て勝敗を判定し、勝者のインスタンスを返すメソッド。

judgeFinalWinner(Player player1, Player player2)は、ゲームの最終的な勝者を判定するメソッドである。引数として、Player 変数の player1、player2 を受け取る。Player1、player2 の勝ち数を Plyer クラスの getWinCount()メソッドで取得。それぞれの勝ち数を比較し最終的な勝者を判定し、勝者を返すメソッド。

startJanken(int round, int doCount, int combo,Player player1, Player player2)メソッドは、ジャンケンを実行するメソッドである。Janken クラスの main メソッドがジャンケン開始を合図し、引数としてラウンド数(round)、試行回数(doCount)、規定連続勝利回数(combo)、プレイヤー(player1,player2)を与える。player1 と player2 の Round 回勝負を doCount 回試行するメソッド。

2.3 ゲームの洗練度

1.1 節で述べたように、一般的に面白いゲームとされるチェスや将棋は洗練度$\sqrt{B}/D$が 0.07 前後である[6]。したがって、洗練度$\sqrt{B}/D$が 0.07 であることは面白いゲームであることの 1 つの基準であると推定される。

本研究では、2.1 節で述べた二種類のラウンドマッチジャンケンの洗練度$\sqrt{B}/D$を求めた。n=2 から n=20 までの値に対し 100,000 回試行した結果を表 1 および表 2 に示す。なお、ジャンケンの自由度 D はグー、チョキ、パーの 3 種なので 3 で固定である。

表 2:R1 の洗練度

勝ち数(n)	2	3	4	5	6	7	8	9	10	11
平均終了ラウンド数(D)	3.74	6.19	8.70	11.32	13,93	16.60	19.31	21.98	24.70	24.47
洗練度 $\sqrt{B}/D$	0.462	0.279	0.199	0.152	0.124	0.104	0.089	0.078	0.070	0.063

勝ち数(n)	12	13	14	15	16	17	18	19	20	
平均終了ラウンド数(D)	30.22	32.97	35.72	38.47	41.26	44.05	46.86	49.67	52.49	
洗練度 $\sqrt{B}/D$	0.057	0.052	0.048	0.045	0.041	0.039	0.036	0.034	0.032	

表 3:R2 の洗練度

勝ち数(n)	2	3	4	5	6	7	8	9	10	11
平均終了ラウンド数(D)	2.00	3.00	4.00	5.00	6.00	7.00	8.00	9.00	10.00	11.00
洗練度 $\sqrt{B}/D$	0.866	0.577	0.433	0.346	0.288	0.247	0.216	0.192	0.173	0.157

勝ち数(n)	12	13	14	15	16	17	18	19	20	24
平均終了ラウンド数(D)	12.00	13.00	14.00	15.00	16.00	17.00	18.00	19.00	20.00	24.00
洗練度 $\sqrt{B}/D$	0.144	0.133	0.123	0.115	0.108	0.101	0.096	0.091	0.086	0.072

表 2 の結果からルール R1 では n=10 のとき洗練度$\sqrt{B}/D$が最も 0.07 に近い事、また n=2 のときが最

も 0.07 から離れているという事がわかる。また、表 3 の結果からルール R2 では n=24 のとき洗練度$\sqrt{B}/D$が最も 0.07 に近づいており、n=2 の時が最も 0.07 から離れている事がわかる。従って、ルール R1 では 10 点先取制にすること、ルール R2 では 24 ラウンドマッチにすることが、最も面白いゲームとなる可能性が高い。

2.4 アンケート結果

洗練度$\sqrt{B}/D$ により得られる値と、ゲームの実際の面白さの関係を検証するために、本研究では 2.3 節の結果をもとに 24 人の被験者を対象に、2 つのアンケートを実施した。

- アンケート 1:ルールは違うが洗練度は近い 2 つを比較(R2(n=24)と R1(n=10))
- アンケート 2:ルールは同じだが洗練度が離れている 2 つを比較(R1(n=2)と R1(n=10))

アンケート 1 では R2(n=24)と R1(n=10)、アンケート 2 では R1(n=2)と R1(n=10)を比較した。
また、それぞれのアンケート実施の目的は洗練度と実際の面白さとの相関の検証だが、より具体的な目的は以下の通りである。

- アンケート 1 の目的:両対象の洗練度が 0.07 に近い場合の実際の面白さとの関係の検証
- アンケート 2 の目的:洗練度が 0.07 に近いものが実際の面白さにつながるのかどうかの検証

各アンケートともに以下のような設問を設けた。

- 設問 1:どちらの方が面白いと感じましたか？
- 設問 2:不満点や改善すべき点はありますか？

以上の条件で実施したアンケートの設問 1 の結果を表 4、表 5 に示す。

表 4:アンケート 1 の結果

選択肢	R2(n=24)	R1(n=10)	いずれでもない
割合	13%	83%	4%

表 5:アンケート 2 の結果

選択肢	R1(n=2)	R1(n=10)	いずれでもない
割合	100%	0%	0%

両アンケートともに「洗練度が 0.07 に近いほうが面白くない」という結果になった。設問 2 ではそう答えた理由として、「時間が長い」、「点差が大きくなるにつれ勝てる見込みがなくなる」といった意見が多くみられた。このことから、求めた洗練度と実際の遊んでもらった時の面白さとの間にある隔たりが「実際にかかる時間」と「点差の肥大化による勝機の減少」にあると思われる。

ここで、難易度 B^D の観点からチェス、将棋と比較し、その関係を検証する。表 6 に今回使用した各ルールの難易度 B^D とチェス、将棋の各ゲームの難易度 B^D[7]を示す。

表 6:アンケート 1,2 で用いたルールのゲームの難易度 B^D

ルール	R2(n=24)	R1(n=2)	R1(n=10)	チェス	将棋
難易度 B^D	$2.8*10^{11}$	60	$7.2*10^{11}$	$3.4*10^{123}$	$1.0*10^{345}$

難易度 B^D の観点から見た場合、以上のような結果になった。チェスや将棋は、今回作成したラウンドマッチジャンケンよりもゲームとして洗練されていると思われる。それは多くのゲームが淘汰されてきた中で、長年生き残っていることからも明らかである。実際の面白さの観点から見てもしかりである。難易度 B^D の観点から見た場合も今回作成した R1(n=2)、R1(n=10)、R2(n=24)の各ルールより、チェスや将棋の値の方がはるかに高い値を示している。それは各局面における自由度 B とゲーム終了までの手数 D に圧倒的な差があるためであると考えられる。つまり極めて複雑なのである。アンケート結果と、各ルールの難易度 B^D の結果を比較したところ、実際の面白さで評価を得た R1(n=2)の著しく低い値を示した。また、アンケートでの評価が著しく低い R2(n=24)、R1(n=10)は R1(n=2)に比べ、高い難易度を示している。このことから、実際に遊んだ時の面白さと難易度 B^D には相関がないと考えられる。

2.5 特別ルールを加えた場合のゲームの洗練度

2.4 節で述べたアンケート結果より、洗練度と実際の面白さとの間にある隔たりが「実際にかかる時間」と「点差の肥大化による勝機の減少」にあると思われる。その 2 点を考慮し、アンケート 1 で得票数が少なかった R2(n=24)に特別ルール R3(m 回連続勝利した場合は即勝利)を追加したものとそうでないものを比較し、「チャンス」を与えた時に洗練度と実際の面白さにどのような変化があるのか検証した。R2(n=24)に特別ルール R3(m)を加え洗練度を求めた結果を表 7 に示す。

表 7: R2(n=24)+R3(m)の洗練度

連続勝利数(m)	2	3	4	5	6	7	8	9	10	11
平均終了ラウンド数(D)	5.80	14.90	20.64	22.98	23.68	23.93	23.97	23.98	24.00	24.00
洗練度 $\sqrt{B}/D$	0.289	0.118	0.084	0.075	0.073	0.072	0.072	0.072	0.072	0.072

連続勝利数(m)	12	13	14	15	16	17	18	19	20	
平均終了ラウンド数(D)	24.00	24.00	24.00	24.00	24.00	24.00	24.00	24.00	24.00	
洗練度 $\sqrt{B}/D$	0.072	0.072	0.072	0.072	0.072	0.072	0.072	0.072	0.072	

表 7 より、m の値が大きくなるほど 0.07 に近づいている事がわかる。ゲームの洗練度の考えによれば、m の増加につれて、より洗練された面白いゲームになっていくが、コンピュータ上で対戦させると、m=10 を超えたあたりから条件特別ルール R3(連続勝利による即勝利)での勝利が見られなくなり、チャンスがチャンスとして機能しなくなるという結果になった。このことから、洗練度$\sqrt{B}/D$はチャンスの多寡を反映していないと言える。

2.6 特別ルールを加えた場合のアンケート結果

2.5 節の結果をもとに、アンケート 3 を実施した。アンケート 3 では、特別ルール R3 を加えたものと、アンケート 1,2 で用いた R2(n=24)というルールを比較する。また、R3 同士の比較を行う目的で二種類の R3(m=3 & m=20)を加えたものを用いた。R2 と R3、そして R3 同士の比較を

行うためである。設問はアンケート 1,2 と同じものを使用した。アンケート 3 の結果を表 8 に示す。なお、アンケート 3 ではアンケート 1,2 と同様に 24 人の被験者を対象とした。

表 8:アンケート 3 の結果

選択肢	R2(n=24)	R2(n=24) & R3(m=3)	R2(n=24) & R3(m=20)	いずれでもない
割合	0%	79%	0%	21%

アンケート 3 の結果、ルール R2(n=24)より特別ルール R3(m=3)を加えたものの方が面白いという結果になった。被験者にその理由を問うと「逆転できる可能性が生まれたから」、「緊張感がある」といったものがあった。また、R3(m=3)と R3(m=20)を比較し、後者が面白くなかった理由として「20 回連続勝利は現実的にありえない」というものがあった。このことから、やはり洗練度がいかに高くともチャンスがなければ面白さは感じないという事がわかる。以上の結果をもう一度、難易度 B^D の観点から検証し実際の面白さと難易度 B^D の相関を検証する。アンケート 3 で用いた 3 種類のラウンドマッチジャンケンの難易度 B^D を算出した結果を表 9 に記す。

表 9:アンケート 3 で用いたルールのゲームの難易度

ルール	R2(n=24)	R2(n=24) & R3(m=3)	R2(n=24) & R3(m=20)
難易度 B^D	$2.8*10^{11}$	$1.1*10^{7}$	$2.8*10^{11}$

アンケートにおいて評価の高かった R2(n=24)&R3(m=3)の難易度 B^D と比べると、R2(n=24)、R2(n=24)&R3(m=20)が同様に高い難易度を示している。この結果から、あらためて難易度と実際の面白さと難易度 B^D の間に相関がないことがわかった。

3 結果・考察

表 4、表 5、表 8 の 3 つのアンケート結果の共通点は、洗練度と実際の面白さが必ずしも相関を持たないこと。またアンケート 3 では、「ゲーム時間に伴い発生する点差」への対抗手段として「チャンス」を与えることにより、より面白く感じられるゲームを作成できることも明らかになった。これらの結果より、洗練度と実際の面白さの間には相関がないこと、「チャンス」を与えることでより面白いゲームの作成が可能であることがわかった。また、難易度 B^D と実際の面白さとの相関を検証した。その結果、実際の面白さと難易度 B^D にも相関はみられなかった。

本研究では、洗練度と実際の面白さとの間にある相関があるのか検証した。結果として、この 2 つの事柄の間には相関がないことがわかった。

4 結論・今後の課題

本研究では、ラウンドマッチジャンケンを題材としてゲームの洗練度と実際の面白さの間に相関があるのか検証した。検証の結果、両者の間には必ずしも相関があるとは限らないことが判明した。一方、「ゲーム時間に伴い発生する点差」への対抗手段としての「チャンス」を考慮することによりゲームを面白くなることが示された。

ゲームを面白くするためには、「チャンス」と「スキル」を考慮する必要がある[1]が、本研究の検証では「チャンス」の付与のみを用いている。したがって、「スキル」をゲームに付加し、実際に遊んだ

時の評価とゲームの洗練度がどのような変化が現れるか検討することで今後の課題である。最終的に比較的高評価であったラウンドマッチジャンケン(R2+R3)でさえ、各ラウンドのジャンケン自体は完全に運任せであった。その原因は各手(グー、チョキ、パー)の差別化ができていないことなどが考えられる。こういった運任せになっている部分を、戦略性を要するルールを追加することで改善し、実際の面白さにどのような変化が発生するのか検証することで、より面白いゲームを作る事が今後の課題である。また、本研究で明らかになった「実際の面白さ」と「難易度 B^D」の関係。今回作成したゲームをより複雑にし、難易度 B^D の値を高めた場合、実際の面白さとゲームの洗練度の値にどのような変化が現れるのかを検証することも今後の課題である。

謝辞

本論文の作成にあたりゼミ教員の石水隆先生には、適切かつ丁寧な指導をいただきました。この場を借りて心より感謝申し上げます。また、アンケート実施の際にご協力いただき、貴重な意見を与えてくださった全ての方に感謝申し上げます。

参考文献

[1] 小森裕介,技術評論社,「なぜ，あなたは Java でオブジェクト指向開発ができないのか」サポートページ,http://gihyo.jp/book/2005/4-7741-2222-X/support

[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 Vol.2005, No.15,pp.138-141 (2005), http://id.nii.ac.jp/1001/00097591/

[3] 飯田弘之：ゲーム研究のいま情報の科学と技術 Vol.62, No.12, pp.527-532 (2012), https://dspace.jaist.ac.jp/dspace/bitstream/10119/10889/1/19097.pdf

[4] 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ゲーム情報学, Vol.2003-GI-010, pp.9-13 (2003), http://id.nii.ac.jp/1001/00058569/

[5] 飯田弘之：コンピュータ囲碁とゲーム情報学，第 7 回情報科学技術フォーラム，(2008), https://www.ipsj.or.jp/10jigyo/fit/fit2008/fit2008program/html/event/pdf/A4_Iida.pdf

[6] 飯田弘之：思考ゲームにおける計算知能の未来，サイエンティフィック・システム研究会 合同分科会 2012 年度会合 「ビッグデータ，どう使う？」，(2012), https://www.ssken.gr.jp/MAINSITE/download/newsletter/2012/20121025-joint/lecture-04/SSKEN_joint2012_iida_summary.pdf

[7] 佐々木 宣介,飯田 弘之：将棋種の歴史的変遷の解析，情報処理学会論文誌 Vol.43, No.10, pp.2990-2997, (2002), http://id.nii.ac.jp/1001/00011455/

[8] 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/

[9] 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,http://www.orsj.or.jp/archive2/or58-03/or58_3_156.pdf

[10] 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

[11] 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.seto.nanzan-u.ac.jp/st/gr-thesis/ms/2006/osaki/03mm015.pdf

[12] 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

付録

以下に、本研究で作成したラウンドマッチジャンケンのプログラムを記載する。第 1 章では各ルール共通のクラスを、第 2 章ではルールごとに違いを持つプログラムをそれぞれ記載する。

1. 共通ルール

「Tactics クラス」

```
// ジャンケン戦略インターフェース。

public interface Tactics {
	//戦略を読み、ジャンケンの手を得る。
	//グー・チョキ・パーのいずれかを Player クラスに定義された 以下の定数で返す。
	//Player.STONE・・・ グー Player.SCISSORS ・・・ チョキ Player.PAPER ・・・ パー

	public int readTactics();
}
```

「RandomTactics クラス」

```
//ランダムに手を決める戦略クラス。
public class RandomTactics implements Tactics {

	public int readTactics() {
		// プレイヤーの手
		int hand = 0;

		// 0 以上 3 未満の小数として乱数を得る
		double randomNum = Math.random() * 3;
		if (randomNum < 1) {
			// randomNum が 0 以上 1 未満の場合、グー
			hand = Player.STONE;
		} else if (randomNum < 2) {
			// randomNum が 1 以上 2 未満の場合、チョキ
			hand = Player.SCISSORS;
		} else if (randomNum < 3) {
			// randomNum が 2 以上 3 未満の場合、パー
			hand = Player.PAPER;
		}

		// 決定した手を戻り値として返す
		return hand;
	}
}
```

2. 各変更ルール

「Judge クラス(R1)」

```
import java.util.ArrayList;

//ジャンケンを行う審判クラス。
public class Judge {
	/**
	 * ジャンケンを開始する。
	 *
	 * @param player1
	 *            判定対象プレイヤー1
	 * @param player2
	 *            判定対象プレイヤー2
	 */
	// 平均ラウンド数のための変数
	double finalRCount = 0;
	double heikinR = 0;
	double sophi = 0;

	void startJanken(int round, int doCount, Player player1, Player player2) {
		// ジャンケンの開始を宣言する
		System.out.println("【ジャンケン開始】");

		// じゃんけんの回数
		int cnt = 0;
		// ラウンド数
		int rCount = 0;

		// ジャンケンをどちらかが 3 回勝つまで行う
		do {
			cnt++;
			rCount++;
			// 平均ラウンド数に加える
			finalRCount++;
			// 何回戦目か表示する
			System.out.println("【" + (cnt) + "回戦目】");

			// プレイヤーの手を見て、どちらが勝ちかを判定する。
			Player winner = judgeJanken(player1, player2);

			if (winner != null) {
				// 勝者を表示する
				System.out.println("" + winner.getName() + "が勝ちました!");

				// 勝ったプレイヤーへ結果を伝える
				winner.notifyResult(true);
```

```
                } else {
                        // 引き分けの場合
                        System.out.println("引き分けです！");
                }
        } while (player1.getWinCount() < round && player2.getWinCount() < round);

        // ジャンケンの終了を宣言する
        System.out.println("【ジャンケン終了】");

        // 最終的な勝者を判定する
        Player finalWinner = judgeFinalWinner(player1, player2);

        // 結果の表示
        System.out.print(player1.getWinCount() + " 対 " + player2.getWinCount()
                        + "で");

        if (finalWinner != null) {
                System.out.println(finalWinner.getName() + "の勝ちです！");
                finalWinner.notifyFinalResult(true);
        } else {
                System.out.println("引き分けです！");
        }

        System.out.println("終了ラウンド数:" + rCount);
        heikinR = finalRCount / doCount;
        System.out.println("平均ラウンド数:" + heikinR);
        sophi = (Math.sqrt(3) / heikinR);
        System.out.println("洗練度:" + sophi);
        player1.setWinCount(0);
        player2.setWinCount(0);
}

/**
 * 「ジャンケン、ポン！」と声をかけ、 プレイヤーの手を見て、どちらが勝ちかを判定する。
 *
 * @param player1
 *            判定対象プレイヤー1
 * @param player2
 *            判定対象プレイヤー2
 * @return 勝ったプレイヤー。引き分けの場合は null を返す。
 */
Player judgeJanken(Player player1, Player player2) {
        Player winner = null;

        // プレイヤー１の手を出す
        int player1hand = player1.showHand();
```

```
                // プレイヤー２の手を出す
                int player2hand = player2.showHand();

                // それぞれの手を表示する
                printHand(player1hand);
                System.out.print(" vs. ");
                printHand(player2hand);
                System.out.println("");

                // プレイヤー１が勝つ場合
                if ((player1hand == Player.STONE && player2hand == Player.SCISSORS)
                                || (player1hand == Player.SCISSORS && player2hand ==
Player.PAPER)
                                || (player1hand == Player.PAPER && player2hand ==
Player.STONE)) {
                        winner = player1;
                }
                // プレイヤー２が勝つ場合
                else if ((player1hand == Player.STONE && player2hand == Player.PAPER)
                                || (player1hand == Player.SCISSORS && player2hand ==
Player.STONE)
                                || (player1hand == Player.PAPER && player2hand ==
Player.SCISSORS)) {
                        winner = player2;
                }

                // どちらでもない場合は引き分け(null を返す)

                return winner;
        }

        /**
         * 最終的な勝者を判定する。
         *
         * @param player1
         *            判定対象プレイヤー2
         * @param player2
         *            判定対象プレイヤー2
         * @return 勝ったプレイヤー。引き分けの場合は null を返す。
         */
        Player judgeFinalWinner(Player player1, Player player2) {
                Player winner = null;

                // Player1 から勝ち数を聞く
                int player1WinCount = player1.getWinCount();

                // Player2 から勝ち数を聞く
                int player2WinCount = player2.getWinCount();
```

```
                if (player1WinCount > player2WinCount) {
                        // プレイヤー1の勝ち
                        winner = player1;
                } else if (player1WinCount < player2WinCount) {
                        // プレイヤー2の勝ち
                        winner = player2;
                }

                // どちらでもない場合は引き分け(nullを返す)

                return winner;
        }

        /**
         * ジャンケンの手を表示する。
         *
         * @param hand
         *            ジャンケンの手
         */
        private void printHand(int hand) {
                switch (hand) {
                case Player.STONE:
                        System.out.print("グー");
                        break;

                case Player.SCISSORS:
                        System.out.print("チョキ");
                        break;

                case Player.PAPER:
                        System.out.print("パー");
                        break;

                default:
                        break;
                }
        }
}
```

「Judge クラス(R2)」

```
//ジャンケンを行う審判クラス。
public class Judge {
        /**
         * ジャンケンを開始する。
         *
         * @param player1
         *            判定対象プレイヤー1
```

```
     * @param player2
     *            判定対象プレイヤー2
     */
    // 平均ラウンド数のための変数
    double finalRCount = 0;

    void startJanken(int round, int doCount, Player player1, Player player2) {
            // ジャンケンの開始を宣言する
            System.out.println("【ジャンケン開始】");

            // じゃんけんの回数
            int cnt = 0;
            // ラウンド数
            int rCount = 0;

            // ジャンケンを n 回行う
            for (int i = 0; i < round; i++) {
                    cnt++;
                    rCount++;
                    // 平均ラウンド数に加える
                    finalRCount++;
                    // 何回戦目か表示する
                    System.out.println("【" + (cnt) + "回戦目】");

                    // プレイヤーの手を見て、どちらが勝ちかを判定する。
                    Player winner = judgeJanken(player1, player2);

                    if (winner != null) {
                            // 勝者を表示する
                            System.out.println("" + winner.getName() + "が勝ちました!");

                            // 勝ったプレイヤーへ結果を伝える
                            winner.notifyResult(true);
                    } else {
                            // 引き分けの場合
                            System.out.println("引き分けです！");
                    }
            }
            // ジャンケンの終了を宣言する
            System.out.println("【ジャンケン終了】");

            // 最終的な勝者を判定する
            Player finalWinner = judgeFinalWinner(player1, player2);

            // 結果の表示
            System.out.print(player1.getWinCount() + " 対 " + player2.getWinCount()
                            + "で");
```

```
                if (finalWinner != null) {
                        System.out.println(finalWinner.getName() + "の勝ちです！");
                        finalWinner.notifyFinalResult(true);
                } else {
                        System.out.println("引き分けです！");
                }

                System.out.println("終了ラウンド数:" + rCount);
                double heikinR = finalRCount / doCount;
                System.out.println("平均ラウンド数:" + heikinR);
                System.out.println("洗練度:" + (Math.sqrt(3) / heikinR));
                System.out.println("ゲームの難易度:" + Math.pow(3, heikinR));
                player1.setWinCount(0);
                player2.setWinCount(0);
        }

        /**
         * 「ジャンケン、ポン！」と声をかけ、 プレイヤーの手を見て、どちらが勝ちかを判定する。
         *
         * @param player1
         *            判定対象プレイヤー1
         * @param player2
         *            判定対象プレイヤー2
         * @return 勝ったプレイヤー。引き分けの場合は null を返す。
         */
        Player judgeJanken(Player player1, Player player2) {
                Player winner = null;

                // プレイヤー１の手を出す
                int player1hand = player1.showHand();

                // プレイヤー２の手を出す
                int player2hand = player2.showHand();

                // それぞれの手を表示する
                printHand(player1hand);
                System.out.print(" vs. ");
                printHand(player2hand);
                System.out.println("");

                // プレイヤー１が勝つ場合
                if ((player1hand == Player.STONE && player2hand == Player.SCISSORS)
                                || (player1hand == Player.SCISSORS && player2hand ==
Player.PAPER)
                                || (player1hand == Player.PAPER && player2hand ==
Player.STONE)) {
```

```
                winner = player1;
            }
            // プレイヤー２が勝つ場合
            else if ((player1hand == Player.STONE && player2hand == Player.PAPER)
                    || (player1hand == Player.SCISSORS && player2hand ==
Player.STONE)
                    || (player1hand == Player.PAPER && player2hand ==
Player.SCISSORS)) {
                winner = player2;
            }

            // どちらでもない場合は引き分け(null を返す)

            return winner;
        }

        /**
         * 最終的な勝者を判定する。
         *
         * @param player1
         *            判定対象プレイヤー2
         * @param player2
         *            判定対象プレイヤー2
         * @return 勝ったプレイヤー。引き分けの場合は null を返す。
         */
        Player judgeFinalWinner(Player player1, Player player2) {
            Player winner = null;

            // Player1 から勝ち数を聞く
            int player1WinCount = player1.getWinCount();

            // Player2 から勝ち数を聞く
            int player2WinCount = player2.getWinCount();

            if (player1WinCount > player2WinCount) {
                // プレイヤー1 の勝ち
                winner = player1;
            } else if (player1WinCount < player2WinCount) {
                // プレイヤー2 の勝ち
                winner = player2;
            }

            // どちらでもない場合は引き分け(null を返す)

            return winner;
        }

        /**
```

```
         * ジャンケンの手を表示する。
         *
         * @param hand
         *            ジャンケンの手
         */
        private void printHand(int hand) {
                switch (hand) {
                case Player.STONE:
                        System.out.print("グー");
                        break;

                case Player.SCISSORS:
                        System.out.print("チョキ");
                        break;

                case Player.PAPER:
                        System.out.print("パー");
                        break;

                default:
                        break;
                }
        }
}
```

「Judge クラス(R3)」

```
// ジャンケンを行う審判クラス。
public class Judge {
        /**
         * ジャンケンを開始する。
         *
         * @param player1
         *            判定対象プレイヤー1
         * @param player2
         *            判定対象プレイヤー2
         */
        // 平均ラウンド数のための変数
        double finalRCount = 0;

        void startJanken(int round, int doCount, int combo, Player player1,
                        Player player2) {
                // ジャンケンの開始を宣言する
                System.out.println("【ジャンケン開始】");

                // じゃんけんの回数
                int cnt = 0;
                // ラウンド数
                int rCount = 0;
```

```
		// ジャンケンを n 回行う
		for (int i = 0; i < round; i++) {
			cnt++;
			rCount++;
			// 平均ラウンド数に加える
			finalRCount++;
			// 何回戦目か表示する
			System.out.println("【" + (cnt) + "回戦目】");

			// プレイヤーの手を見て、どちらが勝ちかを判定する。
			Player winner = judgeJanken(player1, player2);

			if (winner != null) {
				// 勝者を表示する
				System.out.println("" + winner.getName() + "が勝ちました!");

				// 勝ったプレイヤーへ結果を伝える
				winner.notifyResult(true);

			} else {
				// 引き分けの場合
				System.out.println("引き分けです！");
				player1.resetCombo();
				player2.resetCombo();
			}

			// 特別ルール勝敗判断
			if (winner != null) {
				if (winner.getCombo() == combo) {
					break;
				}
			}
		}
		// ジャンケンの終了を宣言する
		System.out.println("【ジャンケン終了】");

		// 最終的な勝者を判定する
		Player finalWinner = judgeFinalWinner(player1, player2, combo);

		// 結果の表示
		System.out.print(player1.getWinCount() + " 対 " + player2.getWinCount()
				+ "で");

		if (finalWinner != null) {
			System.out.println(finalWinner.getName() + "の勝ちです！");
			finalWinner.notifyFinalResult(true);
```

```
            } else {
                    System.out.println("引き分けです！");
            }

            System.out.println("終了ラウンド数:" + rCount);
            double heikinR = finalRCount / doCount;
            System.out.println("平均ラウンド数:" + heikinR);
            System.out.println("洗練度:" + (Math.sqrt(3) / heikinR));
            System.out.println("ゲームの難易度:" + Math.pow(3, heikinR));
            player1.setWinCount(0);
            player2.setWinCount(0);
            player1.resetCombo();
            player2.resetCombo();
    }

    /**
     * 「ジャンケン、ポン！」と声をかけ、 プレイヤーの手を見て、どちらが勝ちかを判定する。
     *
     * @param player1
     *            判定対象プレイヤー1
     * @param player2
     *            判定対象プレイヤー2
     * @return 勝ったプレイヤー。引き分けの場合は null を返す。
     */
    Player judgeJanken(Player player1, Player player2) {
            Player winner = null;

            // プレイヤー１の手を出す
            int player1hand = player1.showHand();

            // プレイヤー２の手を出す
            int player2hand = player2.showHand();

            // それぞれの手を表示する
            printHand(player1hand);
            System.out.print(" vs. ");
            printHand(player2hand);
            System.out.println("");

            // プレイヤー１が勝つ場合
            if ((player1hand == Player.STONE && player2hand == Player.SCISSORS)
                            || (player1hand == Player.SCISSORS && player2hand ==
Player.PAPER)
                            || (player1hand == Player.PAPER && player2hand ==
Player.STONE)) {
                    winner = player1;
                    player1.addCombo();
```

```
                player2.resetCombo();
            }
            // プレイヤー２が勝つ場合
            else if ((player1hand == Player.STONE && player2hand == Player.PAPER)
                    || (player1hand == Player.SCISSORS && player2hand ==
Player.STONE)
                    || (player1hand == Player.PAPER && player2hand ==
Player.SCISSORS)) {
                winner = player2;
                player2.addCombo();
                player1.resetCombo();
            }

            // どちらでもない場合は引き分け(null を返す)

            return winner;
        }

        /**
         * 最終的な勝者を判定する。
         *
         * @param player1
         *            判定対象プレイヤー2
         * @param player2
         *            判定対象プレイヤー2
         * @return 勝ったプレイヤー。引き分けの場合は null を返す。
         */
        Player judgeFinalWinner(Player player1, Player player2, int combo) {
            Player winner = null;

            // Player1 から勝ち数を聞く
            int player1WinCount = player1.getWinCount();

            // Player2 から勝ち数を聞く
            int player2WinCount = player2.getWinCount();

            if (player1WinCount > player2WinCount || player1.getCombo() == combo) {
                // プレイヤー1 の勝ち
                winner = player1;
            } else if (player1WinCount < player2WinCount
                    || player2.getCombo() == combo) {
                // プレイヤー2 の勝ち
                winner = player2;
            }

            // どちらでもない場合は引き分け(null を返す)

            return winner;
```

```
        }

        /**
         * ジャンケンの手を表示する。
         *
         * @param hand
         *            ジャンケンの手
         */
        private void printHand(int hand) {
                switch (hand) {
                case Player.STONE:
                        System.out.print("グー");
                        break;

                case Player.SCISSORS:
                        System.out.print("チョキ");
                        break;

                case Player.PAPER:
                        System.out.print("パー");
                        break;

                default:
                        break;
                }
        }
}
```

「ObjectJanken クラス(R1 & R2)」

```
//ジャンケンプログラム
public class ObjectJanken {

        // 試行回数
        static int testCount = 100000;

        public static void main(String[] args) {

                // 審判のインスタンス生成
                Judge judge = new Judge();

                // プレイヤー１（A さん）の生成
                Player a = new Player("A さん");
```

```
                // A さんに渡す戦略クラスを生成する
                Tactics aTactics = new RandomTactics();

                // A さんに戦略クラスを渡す
                a.setTactics(aTactics);

                // プレイヤー２（B さん）の生成
                Player b = new Player("B さん");

                // B さんに渡す戦略クラスを生成する
                Tactics bTactics = new RandomTactics();

                // B さんに戦略クラスを渡す
                b.setTactics(bTactics);

                // A さんと B さんをプレイヤーとしてジャンケンを開始する

                for (int i = 0; i < testCount; i++) {
                        judge.startJanken(20, testCount, a, b);
                }

                System.out.println("A:" + a.getFinalWinCount() + "対 B:"
                                + b.getFinalWinCount());

        }

}
```

「ObjectJanken クラス(R3)」

```
//ジャンケンプログラム。
public class ObjectJanken {

        // 試行回数
        static int testCount = 100000;
        // ※特別ルール R3(連続勝利規定回数)
        static int combo = 20;

        public static void main(String[] args) {
```

```
                // 審判のインスタンス生成
                Judge judge = new Judge();

                // プレイヤー１（A さん）の生成
                Player a = new Player("A さん");

                // A さんに渡す戦略クラスを生成する
                Tactics aTactics = new RandomTactics();

                // A さんに戦略クラスを渡す
                a.setTactics(aTactics);

                // プレイヤー２（B さん）の生成
                Player b = new Player("B さん");

                // B さんに渡す戦略クラスを生成する
                Tactics bTactics = new RandomTactics();

                // B さんに戦略クラスを渡す
                b.setTactics(bTactics);

                // A さんと B さんをプレイヤーとしてジャンケンを開始する
                for (int i = 0; i < testCount; i++) {
                        judge.startJanken(24, testCount, combo, a, b);
                }
                System.out.println("A:" + a.getFinalWinCount() + "対 B:"
                                + b.getFinalWinCount());

        }

}
```

「Player クラス(R1 & R2)」

```
//ジャンケンを行うプレイヤークラス。
public class Player {
        //グー
        public static final int STONE = 0;
```

```
//チョキ
public static final int SCISSORS = 1;

//パー
public static final int PAPER = 2;

//プレイヤーの名前
private String name;

//プレイヤーの買った回数
private int winCount = 0;

//プレイヤーの最終勝ち数
private int finalWinCount = 0;

//与えられた戦略
private Tactics tactics_;

// プレイヤークラスのコンストラクタ
public Player(String name) {
        this.name = name;
}

//プレイヤーに戦略を渡す
void setTactics(Tactics tactics) {
        tactics_ = tactics;
}

//ジャンケンの手を出す
int showHand() {
        // 与えられた戦略を読んでジャンケンの手を決める
        int hand = tactics_.readTactics();

        // 決定した手を戻り値として返す
        return hand;
}

// 勝敗の結果を受け取り、勝っていた場合は加点する。
```

```
void notifyResult(boolean result) {
        if (true == result) {
                // 勝った場合は勝ち数を加算する
                winCount += 1;
        }
}

//最終的な勝敗の結果を受け取り、加点する。
void notifyFinalResult(boolean result) {
        if (true == result) {
                // 勝った場合は勝ち数を加算する
                setFinalWinCount(getFinalWinCount() + 1);
        }
}

//自分の買った回数を返す
int getWinCount() {
        return winCount;
}

//自分の名前を返す
String getName() {
        return name;
}
//勝ち数を設定する
public void setWinCount(int i) {
        this.winCount = i;
}
//最終勝利回数を返す
public int getFinalWinCount() {
        return finalWinCount;
}
//最終勝利回数を設定する。
public void setFinalWinCount(int finalWinCount) {
        this.finalWinCount = finalWinCount;
}
//勝ち数をリセットする
public void resetWinCount() {
        this.winCount = 0;
```

```
        }
}
```

「Player クラス(R3)」

```
// ジャンケンを行うプレイヤークラス。
public class Player {
        //ぐー
        public static final int STONE = 0;

        //チョキ
        public static final int SCISSORS = 1;

        //パー
        public static final int PAPER = 2;

        //プレイヤーの名前
        private String name;

        //プレイヤーの買った回数
        private int winCount = 0;

        //プレイヤーの最終勝ち数
        private int finalWinCount = 0;

        //与えられた戦略
        private Tactics tactics_;

        // 連続勝利回数用変数
        private int combo;

        // プレイヤークラスのコンストラクタ
        public Player(String name) {
                this.name = name;
                this.combo = 0;
        }

        //プレイヤーに戦略を渡す
        void setTactics(Tactics tactics) {
                tactics_ = tactics;
```

```
}

//ジャンケンの手を出す
int showHand() {
        // 与えられた戦略を読んでジャンケンの手を決める
        int hand = tactics_.readTactics();

        // 決定した手を戻り値として返す
        return hand;
}

// 勝敗の結果を受け取り、勝っていた場合は加点する。
void notifyResult(boolean result) {
        if (true == result) {
                // 勝った場合は勝ち数を加算する
                winCount += 1;
        }
}

//最終的な勝敗の結果を受け取り、加点する
void notifyFinalResult(boolean result) {
        if (true == result) {
                // 勝った場合は勝ち数を加算する
                setFinalWinCount(getFinalWinCount() + 1);
        }

}

//自分の勝った回数を返す
int getWinCount() {
        return winCount;
}

//自分の名前を返す
String getName() {
        return name;
}
//勝ち数を設定する
public void setWinCount(int i) {
```

```
		this.winCount = i;
	}
	//最終勝利回数を返す
	public int getFinalWinCount() {
		return finalWinCount;
	}
	//最終勝利回数を設定する
	public void setFinalWinCount(int finalWinCount) {
		this.finalWinCount = finalWinCount;
	}
	//勝ち数をリセットする
	public void resetWinCount() {
		this.winCount = 0;
	}
	//連続勝利回数を返す。
	public int getCombo() {
		return this.combo;
	}
	//連続勝利回数をリセットする。
	public void resetCombo() {
		if (this.combo != 0) {
			this.combo = 0;
		}
	}
	//連続勝利回数を+1 する。
	public void addCombo() {
		this.combo++;
	}

}
```