公立はこだて未来大学 2018 年度 システム情報科学実習
グループ報告書

Future University Hakodate 2018 System Information Science Practice
Group Report

プロジェクト名

ディーラーをやっつけろ! 複雑系の数理とシミュレーション

Project Name

Beat the Dealer! Mathematics of Complex Systems and Simulation.

グループ名

グループ 1

Group Name

Group 1

プロジェクト番号/Project No.

3

プロジェクトリーダ/Project Leader

1014207　菱田美紗紀　Misaki Hishida

グループリーダ/Group Leader

1016123　薩田凱斗　Kaito Satta

グループメンバ/Group Member

1016007　柏田輝　Hikaru Kashiwada
1016042　尾崎拓海　Takumi Ozaki
1016078　伊藤晋之介　Shinnosuke Ito
1016087　轟木文弥　Fumiya Todoroki
1016118　葛西隼人　Hayato Kasai
1016175　柿崎大輝　Daiki Kakizaki
1016184　鳥谷航大　Koudai Toriya
1016207　渡邊凛　Rin Watanabe
1016231　米村祥裕　Yoshihiro Yonemura

指導教員

川越敏司　川口聡　斎藤朝輝

Advisor

Toshiji Kawagoe　Satoshi Kawaguchi　Asaki Saitou

提出日

2019 年 1 月 16 日

Date of Submission

January 16, 2019

概要

本プロジェクトでは、カジノにおいて最もポピュラーなゲームの一つであるブラックジャックを取り扱っている。ブラックジャックに関する代表的な行動戦略としてベーシックストラテジー、カウンティングという戦略が存在する。プレイヤーはこれらの戦略を用いることでブラックジャックの期待利得を増やすことができた。しかしカジノ側がカウンティングへの対策をしたことで、ブラックジャックにおいてカウンティングを使用することができなくなり、プレイヤーが利得を得ることは難しくなった。そこで我々は既存の戦略と比較し、以下の 2 つの観点から優れている戦略を新たに見つけ出すことをプロジェクトの目標とした。1 つ目の観点はプレイヤーにとって扱いやすく、実行可能な戦略であること、2 つ目は最終的な利得が他の戦略より大きくできることである。

前期では、我々はベーシックストラテジーについてどの程度確かな戦略なのか、他の戦略に比べてどの程度プレイヤーに有利な戦略なのかを検証した。検証ではブラックジャックのシミュレータを作成しシミュレーションを行うことで、各戦略の勝率を調べた。さらに戦略の扱いやすさを判断するために、我々は戦略の複雑性を新しく定義した。そして戦略の勝率と複雑性を合わせた性能指標から戦略の評価を行なった。検証の結果から、複雑性を考慮した場合にはベーシックストラテジーには改善の余地があることがわかった。

後期では前期に定義した複雑性の検証、遺伝的アルゴリズムを用いた新たな戦略の探索、シミュレータの改良を行った。複雑性の検証では実際に戦略を覚えてもらう実験を行った。実験の結果を分析したところ、複雑性が正しい指標であることが確認できた。遺伝的アルゴリズムでは複雑性を含めた性能指標を用いて戦略を探索し、優秀な戦略を得ることができた。シミュレータには賭け金の仕組みを導入することで、勝率だけではなく利得を計算できるようになった。その後、遺伝的アルゴリズムから得られた戦略とベーシックストラテジーを比較するためシミュレーションを行った。シミュレーションでは扱いやすさを考慮するため、プレイヤーが戦略を覚え間違える度合いをエラー率として新たに定義し、戦略間の比較を行った。シミュレーションの結果から遺伝的アルゴリズムから得られた戦略がベーシックストラテジーよりも扱いやすさと利得の両方の点から優れていることが分かった。

キーワード　カジノ, ブラックジャック, ベーシックストラテジー, 複雑性, 遺伝的アルゴリズム

（※文責: 伊藤晋之介）

Abstract

In this project, we deal with Blackjack, one of the most popular casino games. There are "Basic Strategy" and "Counting" as representative behavior strategy concerning Blackjack. Players were able to increase the expected gains of Blackjack by using these strategies. However, as the casino side took countermeasures, Counting could not be used in Blackjack, making it difficult for players to get gains. Therefore, we have set the goal of the project to discover superior strategies from the following two points in comparison with existing strategies. The first point is that it is easy for players to play with a usable strategy, and the second is that the final gain can be larger than other strategies.

In the previous term, we examined how accurate the basic strategy is, and how advantageous it is to players compared with other strategies. In the verification, we created a simulator of Blackjack and conducted a simulation to investigate the winning percentage of each strategy. In addition, to determine how easy it is to handle the strategy, we re-defined the complexity of the strategy.We also evaluated the strategy from the evaluation index which combined the winning percentage and the complexity of the strategy. From the results of the verification, it was found that there is room for improvement in the basic strategy when considering the complexity.

In the latter term, we verified the complexity defined in the previous term, searched for new strategies using Genetic algorithm, and improved the simulator. In the verification of complexity, we conducted an experiment to remember the strategy. From analyzing the results of the experiment, we confirmed that the complexity is the correct index. In the Genetic algorithm, we were able to explore strategies using performance index including complexity and obtained excellent strategies. By introducing a mechanism of betting on the simulator, it became possible to calculate not only the winning percentage but also the gain. Then we conducted a simulation to compare the strategy obtained from Genetic algorithm with the basic strategy. In this simulation, in order to ease handling, the degree to which the player misunderstands the strategy is defined as an error rate, and comparison between the strategies was made. Simulation results show that Genetic algorithm's strategy is superior to basic strategy in terms of both ease of handling and final gain.

Keyword casino, blackjack, basic strategies, complexity, Genetic Algorithm

（※文責: 伊藤晋之介）

目次

第 1 章　研究背景

1.1　ブラックジャックの戦略の歴史

斎藤 (1999) によれば、ブラックジャックの戦略については 1950 年にメリーランド州のとある米国陸軍の研究所に所属していた Roger、Nash、Baldwin らが研究したものが始まりであるといわれている。その後計算機の性能向上により、ブラックジャックのシミュレーションが容易になったことでさらに戦略の研究は進んでいった。ブラックジャックには主に有名な戦略が 2 つ存在する。

1 つ目がベーシックストラテジーと呼ばれる戦略だ。ベーシックストラテジーはディーラーのアップカードと自分の手札の状況によってプレイヤーが選択するべき最適な行動を決定するという戦略である。

2 つ目はカウンティングと呼ばれる戦略だ。カウンティングはブラックジャックのゲーム中で既に使われたカードを記憶することで、プレイヤーが有利になるように自分がとるべき戦略を決定していくというものである。

前期ではベーシックストラテジーについて調査、検証をした。後期ではカウンティングについて詳しく調査し、シミュレーションを行った。なおカウンティングについてはカジノ側に対策をされており、この戦略を使用していることがカジノ側に気づかれた場合、プレイヤーはカジノから退場させられることもあるなど、現在カウンティングをカジノで扱うことは難しくなっている。

（※文責: 伊藤晋之介）

1.2　ブラックジャックのルール

トランプの扱いについて

- ジョーカーはゲームでは使用しない
- 52 枚を 1 デックとして使用する
- 2〜10 のカードは書いてある数字の通りに扱う
- J・Q・K は 10 として扱う
- A は 11 または 1 で都合の良いほうとして扱う

勝敗の決定について

- 手札の合計値が 21 以下で、合計値が大きい方の勝利
- ディーラーとプレイヤーの手札の合計値が同じ場合は引き分けとなる
- 手札の合計値が 21 を超えることをバーストという
- バーストしたプレイヤーはその時点で負けとなる

Beat the Dealer! Mathematics of Complex Systems and Simulation.

- ディーラーがバーストした場合はすべてのプレイヤーが勝ちとなる
- プレイヤーとディーラーの両方がバーストしている場合、ディーラーの勝ちとなる
- 最初の手札の合計値が 21 の場合ブラックジャックといい、最も強い手札となる

賭け金の扱い

- プレイヤーが勝つと賭け金の 2 倍が払い戻される
- プレイヤーがブラックジャックで勝つと賭け金の 2.5 倍が払い戻される
- ディーラーが勝つと賭け金を没収される
- 引き分けの場合賭け金は賭けたプレイヤーに払い戻される

プレイヤーの選択肢

- ヒット：山札からカードを 1 枚手札に追加すること 21 を超えない限り、何度でもできる。
- スタンド：カードを引かずに今の手札で勝負すること。
- サレンダー：負けを認めることで、賭け金の半分をもらうことができる。最初の行動でのみ使える。
- ダブルダウン：賭け金を 2 倍にして 1 度だけヒットをし、その後スタンドする。最初の行動でのみ使える。
- スプリット：最初に配られた 2 枚のカードが同じ数字だった場合使用可能。最初の賭け金と同じ金額を追加して、それらを 2 つに分割して、それぞれで勝負することができる。
- インシュランス：ディーラーの表向きのカード（アップカード）が「A」の場合使える。最初の賭け金の半分を使い、ディーラーがナチュラルブラックジャックになればその賭け金の２倍が払い戻される。自分の手札がブラックジャックである場合に行うインシュランスのことをイーブンマネーと呼ぶ。

ただし前期の活動ではプレイヤーの行動はヒットまたはスタンドに限定している。

（※文責: 柏田輝）

1.2.1 ゲームの流れ

まずデックをシャッフルし、カットカードと呼ばれるカードをランダムにデックに入れる。カットカードが近づくとデックがシャッフルされる。ゲーム開始時に各プレイヤーは賭け金をテーブルに置く。その後ディーラーは自身と全てのプレイヤーにカードを 2 枚ずつ配る。この時ディーラーのカードは 1 枚を裏向きに、もう 1 枚を表向きにして配る。ディーラーの表向きのカードをアップカードという。カードを配り終えるとプレイヤーの行動に移る。プレイヤーがスタンドもしくはバーストするかイーブンマネーだった場合、プレイヤーの行動は終了となる。全てのプレイヤーの行動が終了すると、ディーラーの行動に移る。ディーラーがスタンドもしくはバーストした場合、ディーラーの行動は終了となる。その後ディーラーは各プレイヤーと勝敗を確認し、それに応じた支払いが行われてゲームが終了する。

（※文責: 柏田輝）

1.2.2　ディーラーの行動

ディーラーの行動は常に一定となっている。ディーラーは手札の合計が 17 以上になるまでヒットを続けなければならないとルールで定められている。そのためディーラーの手札の合計値は最終的に 17,18,19,20,21, バーストのいずれかとなる。

（※文責: 柏田輝）

1.3　ブラックジャックにおける従来の戦略

1.3.1　従来の戦略

ブラックジャックにおける主な既存の戦略はベーシックストラテジーとカウンティングである。

ベーシックストラテジーは一回ごとのゲームを想定しており、自分の手札とアップカードのみによって勝率が高い行動を決定する戦略である。そのため、行動を決定するまでに使用されたカードは行動決定に影響せず、残りのカードの予測も行わない。

また、カウンティングは、デックをシャッフルしない状態で、カードを使い続けた場合のゲームに対して発生する利得を最大にすることを目的とした戦略である。ゲームで使われたカードを記憶し、残りのカードを予測する。そこから自分の有利・不利を決定し賭け金の増減を行うという戦略である。

（※文責: 鳥谷航大）

1.3.2　ベーシックストラテジー

ブラックジャックの最も有名な既存の戦略としてベーシックストラテジーという戦略が挙げられる。ベーシックストラテジーは Thorp(1962) が発表した戦略である。元となるアイデアとして、Roger(1956) の確率計算が利用されている。

先述したとおり、ベーシックストラテジーでは、残りのカードの予測を行わない。そのため、後述で詳しく説明するが、計算に用いる確率は仮定に基づき簡略化されたものであり、それによって戦略が決定される。

（※文責: 鳥谷航大）

1.3.3　ベーシックストラテジーの導出

ベーシックストラテジーは以下の前提条件を持つ。

- 前提条件

使用されているデック数は無限である

つまり、場にカードが何枚使われていても、どのカードを引く確率も常に $\frac{1}{13}$ であるとし、この前提条件の元で、アップカードと手札の合計値からヒットした時とスタンドした時の勝率から導出される。具体的には以下のような手順によって導出する。

1. プレイヤーの手札の合計値とアップカードの組み合わせごとにヒットした場合の勝率とスタンドした場合の勝率を求める
2. ヒットした場合の勝率からスタンドした場合の勝率を引き、その差を求める
3. 求めた差が 0 以上であった場合はヒット、0 未満であった場合はスタンドが有効であるとする

ここで、ベーシックストラテジーの導出に移る前に以下のような定義を行う。

x: プレイヤーに最初に配られた 2 枚のカードの合計値 ($x \leq 21$)
D: アップカード
M(D): ディーラーのアップカードによるプレイヤーがスタンドできる最低の手札の合計値
J: プレイヤーが 1 回ヒットした後の最終的な手札の合計値
T: ディーラーの最終的な手札の合計値 ($T \geq 17$)
$E_{d,x}$: プレイヤーの手札の合計値が x の時にヒットした場合の勝率
$E_{s,x}$: プレイヤーの手札の合計値が x の時にスタンドした場合の勝率
$P(Y)$: 式 Y が成立する確率

まず $E_{s,x}$ について考える。$E_{s,x}$ は、$T > 21$ または $T < x$ の時、プレイヤーは x でスタンドした場合、そのゲームに勝利し、$x < T \leq 21$ の時、プレイヤーは x でスタンドした場合、そのゲームに敗北する。また、$T = x$ の時は引き分けとなり、x でスタンドした場合利得は 0 である。これらのことから $E_{s,x}$ は以下のような式によって表せる。

$$\begin{aligned} E_{s,x} =& P(T > 21) + P(T < x) - P(x < T \leq 21) \\ =& 2P(T > 21) + 2P(T < x) + P(T = x) - 1 \end{aligned}$$

次に $E_{d,x}$ について考える。$E_{d,x}$ は、手札の合計値が x でヒットした場合は以下の 3 つの場合に分けられる。

1. $J < 17$

   T は常に 17 以上であるから、この時にプレイヤーが勝利する期待値は以下のように表せる。

   $$P(T > 21) - (1 - P(T > 21)) = 2P(T > 21) - 1$$

2. $17 \leq J \leq 21$

   同様にこの時にプレイヤーが勝利する期待値は、

   $$P(T > 21) + P(T < J) - P(J < T \leq 21)$$

   となる。

3. $J > 21$

この時、T がどのような値でもプレイヤーは敗北する。つまり期待値は -1 となる。

以上のことから $E_{d,x}$ は、

$$\begin{aligned} E_{d,x} =& P(J < 17)(2P(T > 21) - 1) - P(J > 21) \\ &+ \sum_{j=17}^{21} P(J = j)(P(T > 21) + P(T < j) - P(j < T \leq 21)) \end{aligned}$$

となる。ここで $E_{d,x}$ の第 3 項の部分に注目する。第 3 項は、T が常に 17 以上 ($T \leq 17$) であること、J が 17 以上 21 以下 ($17 \leq J \leq 21$) であることから以下のような変形を行うことができる。

$$\begin{aligned} &\sum_{j=17}^{21} P(J = j)(P(T > 21) + P(T < j) - P(j < T \leq 21) \\ &= \sum_{j=17}^{21} P(J = j)(P(T > 21) + P(T < j) - (1 - P(T > 21) - P(T < j) - P(T = j)) \\ &= \sum_{j=17(}^{21} P(J = j)(2P(T > 21) - 1 + 2P(T < j) + P(T = j)) \\ &= (2P(T > 21) - 1) \sum_{j=17(}^{21} P(J = j) + 2 \sum_{j=17(}^{21} P(J = j)P(T < j) + \sum_{j=17(}^{21} P(J = j)P(T = j) \\ &= (2P(T > 21) - 1) \sum_{j=17(}^{21} P(J = j) + 2P(T < J \leq 21) + P(T = J \leq 21) \end{aligned}$$

また、$P(J < 17) + \sum_{j=17}^{21} P(J = j) = P(J \leq 21)$ と書けるため、$E_{d,x}$ は、

$$\begin{aligned} E_{d,x} &= P(J < 17)(2P(T > 21) - 1) - P(J > 21) \\ &\quad + (2P(T > 21) - 1) \sum_{j=17(}^{21} P(J = j) + 2P(T < J \leq 21) + P(T = J \leq 21) \\ &= (2P(T > 21) - 1)(P(J < 17) + \sum_{j=17}^{21} P(J = j)) - P(J > 21) + 2P(T < J \leq 21) + P(T = J \leq 21) \\ &= (2P(T > 21) - 1)P(J \leq 21) - P(J > 21) + 2P(T < J \leq 21) + P(T = J \leq 21) \\ &= (2P(T > 21) - 1)(1 - P(J > 21)) - P(J > 21) + 2P(T < J \leq 21) + P(T = H \leq 21) \end{aligned}$$

となる。したがって、行動決定式 $E_{d,x} - E_{s,x}$ は、

$$\begin{aligned} E_{d,x} - E_{s,x} &= (2P(T > 21) - 1)(1 - P(J > 21)) - P(J > 21) + 2P(T < J \leq 21) + P(T = H \leq 21) \\ &\quad - (2P(T > 21) + 2P(T < x) + P(T = x) - 1) \\ &= -2P(T < x) - P(T = x) - 2P(T > 21)P(J > 21) + 2P(T < J \leq 21) + P(T = J \leq 21) \end{aligned}$$

となる。ベーシックストラテジーの導出を行う上で、x について以下の 3 つの場合分けを行う。

$$x < 12 \tag{1.1}$$
$$12 \leq x \leq 16 \tag{1.2}$$
$$x > 16 \tag{1.3}$$

$x < 12$ のとき、$T \geq 17$ であるから、$P(T < x)$ と $P(T = x)$ は 0 である。同様に、$P(J > 21)$ も 0 であるから、$E_{d,x} - E_{s,x}$ は常に 0 以上 $(E_{d,x} - E_{s,x} \geq 0)$ となる。したがって、$x < 12$ において、プレイヤーの行動は全てヒットとなる。

$12 \leq x \leq 16$ のとき、先述と同様に $P(T < x)$ と $P(T = x)$ は 0 である。この時の行動決定式 $E_{d,x} - E_{s,x}$ は以下のように表すことができる。

$$E_{d,x} - E_{s,x} = -2P(T > 21)P(J > 21) + \sum_{t-17}^{21} P(T = t)(2P(t < J \leq 21) + P(J = t))$$

また、プレイヤーが引くカードの確率は全て同じであるため、

$$P(J - x = 10) = \frac{4}{13}$$
$$P(J - x = i) = \frac{1}{13} \quad i = 2, 3, ..., 9, (1, 11)$$

となる。したがって、

$$P(J > 21) = \frac{1}{13}(x - 8)$$
$$P(T < J \leq 21) = \frac{1}{13}(21 - T)$$
$$P(J = T) = \frac{1}{13}$$

とそれぞれ表すことができるため、$E_{d,x} - E_{s,x}$ は以下のように表すことができる。

$$E_{d,x} - E_{s,x} = -\frac{2}{13}(x - 8)P(T > 21) + \frac{1}{13}\sum_{t-17}^{21} P(T = t)(43 - 2t)$$

ここで $x = x_0$ として、$E_{d,x_0} - E_{s,x_0} = 0$ の時を考える。この時 x_0 は、

$$x_0 = 8 + \frac{1}{2}\frac{\sum_{t=17}^{21}(43 - 2t)P(T = t)}{P(T > 21)}$$

となる。このことから、$12 \leq x \leq 16$ のとき、$M(D) = [x_0] + 1$ となる。

$x > 16$ のとき、まず、$x = 17$ を考える。この時 $E_{d,x} - E_{s,x}$ は、

$$E_{d,x} - E_{s,x} = -\frac{18}{13}P(T > 21) - \frac{5}{13}P(T = 17) + \frac{1}{13}\sum_{t=18}^{21}(43 - 2t)P(T = t)$$

となる。この時 $E_{d,x} - E_{s,x}$ は、全ての D に対して常に 0 未満となる。つまり、$E_{d,x} - E_{s,x} < 0$ であるから、$M(D) \leq 16$ となる。

以上のことから、ベーシックストラテジーは表 1.1 となる。この戦略表は、縦軸がプレイヤーの手札の合計値を表し、横軸がディーラーのアップカードを表す。それぞれの交わる場所がプレイ

ヤーの取る最適な行動となっていて、S がスタンド、H がヒットを表す。例えば、プレイヤーの手札の合計値が 12、ディーラーのアップカードが 2 であった場合、プレイヤーの取るべき行動は H、つまりヒットとなる。

表 1.1　ベーシックストラテジーの戦略表

		ディーラーのアップカード									
		2	3	4	5	6	7	8	9	10	A
手札の合計	19 以上	S	S	S	S	S	S	S	S	S	S
	18	S	S	S	S	S	S	S	S	S	S
	17	S	S	S	S	S	S	S	S	S	S
	16	S	S	S	S	S	H	H	H	H	H
	15	S	S	S	S	S	H	H	H	H	H
	13〜14	S	S	S	S	S	H	H	H	H	H
	12	H	H	S	S	S	H	H	H	H	H
	11 以下	H	H	H	H	H	H	H	H	H	H

（※文責: 鳥谷航大）

1.4　カウンティングについて

カウンティングとは、ブラックジャックにおいてゲーム内で使用されているデックの状態がプレイヤーにとって有利な状態か、あるいは不利な状況であるかを見極めるための高等テクニックである。プレイヤーにとって有利な時には、リスクを最小限に抑えつつ、より多くの金額を賭けることができる。また、カウンティングを使用することによってデック内に残っているカードの構成に基づいて、プレイヤーは自身の行動選択を変更させることも可能である。Thorp(1962) が最初に発明して以来そのカウンティングの手法ついては数多くの種類が誕生してきたが、本プロジェクトでは High-Low 法と KO 法と呼ばれる 2 種類のカウンティング手法を使用してシミュレーションなどを行った。ここでは、その High-Low 法と KO 法の 2 つについて説明する。

（※文責: 薩田凱斗）

1.4.1　High-Low 法

まずは High-Low 法について説明する。High-Low 法の特徴はランニングカウントと同じ分のユニットを賭けるということである。ランニングカウントとは、各カードに割り当てられた値の暫定的な合計値のことである。自分自身の手札の値以外にも、他のプレイヤーのカードの値とディーラーのアップカードの値も含む。そして、ユニットとは自分が決めるベット額の単位のことである。一般的には、ミニマムベットを基準にする場合が多い。例えば、あるカジノのブラックジャックのテーブルでのミニマムベットが 5 ドルだとすると、ユニットも 5 ドルに設定する。また、ランニングカウントを複数デックで使用する場合には、トゥルーカウントというものを算出する必要が

ある。トゥルーカウントは、ランニングカウントを残りデック数で割ったものである。例えば、ランニングカウントが 10 で残りデック数が 5 の時、トゥルーカウントは 10 割る 2 で 5 となる。では、なぜトゥルーカウントを使用する必要があるのかを説明する。仮にランニングカウントが +5 で残りデック数が 6 の場合とランニングカウントが +5 で残りデック数が 1 の場合は全く違った状況となる。前者はデックごとに 1 枚のハイカードが残っているため、プレイヤーにとっては有利な状況とは言えない。後者は、残り 1 デック、つまり 52 枚のうち 10 と A が 5 枚残っておりプレイヤーにとって有利な状況であることがわかる。High-Low 法のカウント方法は以下の表 1.2 のようにカードの値が 2 から 6 の時にはランニングカウントをプラス 1、7 から 9 の時はランニングカウントを 0、そして 10、A の時にはマイナス 1 をランニングカウントに加えていくというものである。ただし、Thorp(1962) が High-Low 法を発表したときは使用されていなかった。そのため今回はトゥルーカウントを使用しないでシミュレーションを行った。また、High-Low 法においては、初期ランニングカウントを 0 から開始する。

表 1.2　High-Low 法でのカードカウント

2	3	4	5	6	7	8	9	10	A
+1	+1	+1	+1	+1	0	0	0	-1	-1

このランニングカウントがプラスの場合は、10 や A などのカードがデック内に残っているためプレイヤーにとって手札の合計が 21 となるブラックジャックとなる可能性が高く、非常に有利になるためベッド額を増やしていき、反対にランニングカウントがマイナスの時は不利な状況なためベッド額を抑えるといったものである。また、以下の表 1.3 は High-Low 法を使用した際のベットシステムである。

表 1.3　High-Low 法でのベットシステム

ランニングカウント	1 以下	2	3	4	5	6	7	8	9	10 以上
ベット (ユニット)	1	2	3	4	5	6	7	8	9	10

先述した通り、High-Low 法はカウント値と同じユニットを賭けていく。

（※文責: 薩田凱斗）

1.4.2　KO 法

次に KO 法についてである。KO 法は Vancura(1998) が発表したカウンティング手法である。以下の表 1.4 のようにカードの値が 2 から 7 の時にはランニングカウントをプラス 1、8 と 9 の時はランニングカウントを 0、そして 10、A の時にはマイナス 1 をランニングカウントに加えていくというものである。High-Low 法との唯一の違いは 7 をローカードとして扱っているため、カウント値が 0 ではなく +1 となっている。そのため、それぞれのカウント値を足し合わせると +4 となる。これはアンバランスシステムとも呼ばれ、High-Low 法のようにトゥルーカウントを算出する必要がない。

表 1.4　KO 法でのカードカウント

2	3	4	5	6	7	8	9	10	A
+1	+1	+1	+1	+1	+1	0	0	-1	-1

KO 法の大きな特徴としてはデック数に応じて、初期ランニングカウントを変えていくということである。カジノで使用されるデック数で一般的な 6 デックを例にとると、初期ランニングカウントは-20 である。仮に 6 デックの場合には合計で 24 枚の A があり、そこから 1 デック分の A、つまり 4 枚を抜いた数を初期ランニングカウントとして扱う。なので 1 デックの場合の初期ランニングカウントは 0、同様に 2 デックの場合は-4、6 デックの場合は-20 が初期ランニングカウントとなる。以下の表 1.5 は KO 法を使用した際のベットシステムである。

表 1.5　KO 法でのベットシステム

ランニングカウント	-5 以下	-4	-3	-2	-1	0	+1	+2	+3	+4 以上
ベット (ユニット)	1	2	2	3	4	5	6	8	9	10

ランニングカウントが-4 と-3 の時にはユニットを 2 にし、それ以降はユニットを 1 つずつ上げていくが、ランニングカウントが +1 から +2 になるときはユニットを 2 つ増やす。

（※文責: 薩田凱斗）

1.5　ベッティングシステムの検討

カウンティング時の適切なベッティングシステムが存在しないため、カウンティングに適用可能な手法を調査した。その結果ベッティングシステムは以下のようなグループに分けることができた。

- 勝ち負けに応じて一定の倍率で賭け額を変化させる手法
- 数列を用意し、数列を操作しながら賭け額を変化させる手法
- 資金の何割かを賭ける手法
- 勝率に応じて賭け額を変化させる手法
- その他の手法

以上の 5 つについてこれから説明する。

（※文責: 柏田輝）

1.5.1　勝ち負けに応じて一定の倍率で賭け額を変化させる手法

勝ち負けに応じて賭け額を変化させる手法には以下のような手法がある。

- マーチンゲール法

- グランマーチンゲール法
- パーレー法
- グランパーレー法

マーチンゲール法

マーチンゲール法とは、基準となる賭け金 (ユニット) を決めて、負けるたびにユニットを倍に増やしていき、勝った時にユニット数を 1 に戻すという手法である。例えば、1 ユニットを 100 と設定した場合、1 度負けると 100 × 2 となり 200、2 度負けると 200 × 2 となり 400 となる。この手法の特徴としては、1 度勝てば負け額をすべて取り戻し、1 ユニット分だけ利益を出すことが出来る。しかし、負け続けると賭けるユニット数が指数関数的に増えてしまい、破産しやすいというデメリットや、カジノでは賭け金の上限が設定されているため、上限に達すると使用できないといった問題点がある。

グランマーチンゲール法

グランマーチンゲール法とは、負けるたびにユニットを 2 倍しその値に 1 を加えて勝つまでユニット数を増やしていき、勝った時にユニット数を 1 に戻す手法である。例えば、1 ユニットを 100 と設定した場合、1 度負けると 100 × 2+100 となり 300、2 度負けると 300 × 2+100 となり 700 となる。この手法の特徴として、最初の 1 ユニット目を賭けてから勝った場合までのゲーム回数× 1 ユニット分の利益を出すことができる。つまり、マーチンゲール法をハイリスクハイリターンにした手法であるため、勝ち続けることができれば利益を出すことができるが、負け続けるとマーチンゲール法より早く賭け金の上限に達してしまったり、より破産しやすくなってしまう。

パーレー法

パーレー法とは、勝つたびにユニット数を 2 倍にする手法である。例えば、1 ユニットを 100 と設定すると、1 度勝つと 100 × 2 となり 200、2 度勝つと 200 × 2 で 400 となる。この手法の特徴として、連勝することで少ない賭け金で大きな利益を出すことができる。しかし、1 度でも負けてしまうと利益がマイナスになってしまうのである程度利益を得た後、自分でユニット数を 1 に戻す必要がある。

グランパーレー法

グランパーレー法とは、勝つたびにユニット数を 2 倍にしその値に 1 を加えていく手法である。例えば、1 ユニットを 100 と設定した場合の賭け金は、1 度勝つと 100 × 2+100 となり 300、2 度勝つと 300 × 2+100 となり 700 となる。この手法の特徴として、勝ち続けると利益を指数関数的に増やすことができる。しかし、負けた場合に最初の 1 ユニット目を賭けてからその時までのゲーム回数× 1 ユニット分負けてしまう。つまり、パーレー法をハイリスクハイリターンにした手法である。

（※文責: 柏田輝）

1.5.2　数列を用意し、その数列を操作しながら賭け額を変化させる手法

数列を用意し、その数列を操作しながら賭け額を変化させる手法には以下のような手法がある。

- 2in1 法
- モンテカルロ法
- バーネット法

2in1 法

2in1 法とは、数列の両端を足した数のユニット数を賭ける手法である。この手法は 2 連敗した後に適用され、負ける度にその直前に賭けたユニット数を右端に記録していき、勝つ度に記録の両端の数字を 1 つずつ削除する。毎ゲーム時に賭けるユニット数は、勝ち負けに関わらず記録の両端の数字を合計したユニット数を賭ける。例えば、2 連続で負けた後の数列は 1,1 となり、ここから適用する。両端数字が 1,1 なので次は 2 ユニット賭ける。ここで負けた場合数列が 1,1,2 となり、次は 3 ユニット賭けることになる。更ににここで負けた場合数列が 1,1,2,3 となり次は 4 ユニット賭けることになる。ここで勝つと数列の両端の数字である 1,3 を削除し数列が 1,2 に変化するので、次は 3 ユニット賭けることになる。そこでまた勝つことにより、数列の両端の数字がなくなり、負けた分を全て回収できたことになる。この手法の特徴は、2 回分の負け額を 1 度の賭けで回収のすることができる。また、マーチンゲール法ほど賭け額が増えないので破産率が低いのも特徴である。しかし、この手法ではマーチンゲール法のように 1 度勝っただけでは利益が増えない。つまり、連勝しなければ利益を得ることができない手法となっている。また実際のカジノでは、メモを見たりメモを取ることができないため、数列を暗記し、ややこしい計算を頭の中で行わなければならないといったデメリットもある。

モンテカルロ法

モンテカルロ法とは、これは最初の数列を 1,2,3 とし、数列の両端を足した数だけユニット数を賭ける手法である。負ける度に、数列に直前にその直前に賭けたユニット数を付け加え、勝つ度に配当が 2 倍のゲームでは記録の両端の数字を 1 つずつ削除し、配当が 3 倍のゲームでは両端の数字を 2 つずつ削除する。毎ゲーム時に賭けるユニット数は、勝ち負けに関わらず記録の両端の数字を合計したユニット数を賭ける。例えば、配当が 2 倍のゲームでは、1 度目は 1,2,3 となっているので両端数字が 1,3 であり、4 ユニット賭ける。この後負けた場合に、数列は 1,2,3,4 となり、両端数字が 1,4 となるので、次は 5 ユニット賭ける。ここで勝った場合は、両端数字の 1,4 を削除するので数列が 2,3 になるので次に賭けるユニット数は両端数字 2,3 なので 5 ユニット賭けることになる。ここで勝つと両端数字である。2,3 が削除されるので、次に賭けるときは再び 1,2,3 の数列を使う。また配当が 3 倍のゲームでは、1 度目は 1,2,3 となっているので両端数字が 1,3 であり、4 ユニット賭ける。この後負けた場合に、数列は 1,2,3,4 となり、両端数字が 1,4 となるので、次は 5 ユニット賭ける。ここで負けた場合、数列が 1,2,3,4,5 となりり次に賭けるユニット数は両端数字が 1,5 なので 6 になる。ここで勝った場合、両端数字を 2 つずつ削除するので数列は 3 となり、数

列の要素数が 2 個以下のため、次に賭けるときは再び 1,2,3 の数列を使う。特徴としては、2in1 法と同じように、数列を利用する関係上カジノでは扱いにくい。また、この手法は配当が 3 倍のゲームでは儲かるが、2 倍ゲームでは必ずしも利益が出て終わる手法ではないのでブラックジャックには向いていない。

バーネット法

バーネット法とは、賭けるユニット数を 1,3,2,6 のように変化させる手法である。最初に 1 ユニット賭け勝った場合に 3,2,6 のように賭けていく手法で、6 ユニット賭けた後に勝った場合は、負けるまで 6 ユニットを賭け続け、負けた場合は数列の最初の 1 からまたかけ始めていく手法である。この手法の特徴として、連勝時には賭け金を上げて利益を得る。連敗時には賭け金を下げてリスクを減らすことができる。連勝や連敗に対しては非常に頼もしいが、勝ち負けが交互となったり、短いスパンで勝ち負けが同数で進行した場合は、効果を発揮することができない。

（※文責: 柏田輝）

1.5.3 資金の何割かを賭ける手法

資金の何割かを賭ける手法には以下のような手法がある。

- 10 %投資法
- 全額投資法

10% 投資法

10% 投資法とは、自分がゲーム使える額全体のうちから 1 度のゲームにつき、その 10% を賭けるという手法である。この手法の特徴としては、絶対に破産しない上に計算が単純なので覚える必要がない。また、連敗すると賭け金が低くなるので、負けやすい間の不利益を抑えることができ、連勝すると賭け金が高くなるので、勝ちやすい間の利益が高くなる。しかし、今までのベッティング手法とは違い、負け額を確実に回収したり、利益を確実に出す手法ではないので、利益を得にくいといったデメリットがある。また、勝率が 50% を下回ってしまうと持ち金が 0 に収束していくというデメリットもある。

全額投資法

全額投資法とは、自分がゲーム使える額全体を 1 度のゲームですべて賭ける手法である。この手法の特徴としては、絶対に破産はしないが、1 度負けると賭けることのできる金額を全て失うことになる。しかし、カジノでは基本的に長期的に勝負し続けると、少しずつプレイヤー側の利益がマイナスになることが多いのでその裏をかくことができる手法である。

（※文責: 柏田輝）

1.5.4　勝率に応じて賭け額を変化させる手法

勝率に応じて賭け額を変化させる手法には以下のような手法がある。

- ケリー基準
- ハーフケリー

ケリー基準

ケリー基準とは、以下のような公式から賭け額を決定する。

$$(AP-Q)/A$$

A は勝った場合に帰ってくる配当 (デシマルオッズ) から 1 引いた値、つまり勝った場合の純粋な利益であり、P は勝つ確率であり、Q は負ける確率である。これらを計算し出た値に 1 を足した値を賭け額のユニットとなる。例えば、基準となる賭け額が 100、勝つ確率が 52%、負ける確率が 48%、デシマルオッズが 2 だった場合は、(1*0.52-0.48)/1=0.04 となるので、賭け額は 104 となる。

ハーフケリー

ハーフケリーとは、以下のような公式をから賭け額を決定する。

$$(AP-Q)/A/2$$

つまりケリー基準の出た値を半分にし、賭け額を決定する手法である。例えば、基準となる賭け額が 100、勝つ確率が 52%、負ける確率が 48%、デシマルオッズが 2 だった場合は、(1*0.52-0.48)/1=0.04 となり、その値の半分は 0.02 で、賭け額は 102 となる。つまり、ハーフケリーはケリー基準をローリスクローリターンにした手法であるといえる。これらの手法の特徴としては、他のベッティングシステムとは異なり勝率を参照するので、カウンティングを併用しやすいといったメリットがある。一方で、毎回勝率を計算しなければならないといったデメリットがある。

（※文責: 柏田輝）

1.5.5　その他の手法

その他の方法には以下のような手法がある。

- ダランベール法
- 逆ダランベール法
- 31 システム
- ココモ法

ダランベール法

ダランベール法とは、勝負に負けたときにユニットを 1 増やし、勝ったときに 1 減らすという手法である。例えば、1 ユニット 100 と設定した場合、1 度負ければ 100+100 となり 200、2 度負ければ 200+100 となり 300、1 度勝てば 300-100 となり 200 となる。この手法は見た目の通りとても単純であり覚えやすいという特徴がある。また、勝っているときでも負けているときでも、利得の増減がゆるやかである。連敗しても長い目で見ればある程度マイナスになった分の利得を取り戻しやすいというメリットがある。しかし、勝ちと負けがが交互に繰り返された場合には効果がみられないというデメリットがある。この手法は、勝率が約 50 %であり、勝利した際の払い戻しが賭け金の 2 倍であるゲームで有効である。つまりブラックジャックで有効である。

逆ダランベール法

逆ダランベール法とは、勝負に勝ったときにユニットを 1 増やし、負けたときに 1 減らすという、ダランベール法と真逆の手法である。例えば、1 ユニットを 100 と設定した場合、1 度勝てば 100+100 となり 200、2 度勝てば 200+100 となり 300、1 度負ければ 300-100 となり 200 となる。ダランベール法と同じで、とても単純であり覚えやすいという特徴がある。また、連勝すれば急激に利得が増えるというメリットがある。逆に、連勝しなければ大きな利得を得ることができず、ダランベール法と同じく、勝ちと負けが繰り返された場合では利得がでにくいというデメリットがある。つまり、大勝ちはできないが逆に大負けもしなく、連敗してもダランベール法のように賭け金が膨大にならず、賭け金の上限に引っかかることもない。

31 システム

21 システムとは、負けた時にユニットを 1，1，1，2，2，4，4，8，8 の順番で増やし、勝ったときには現在の賭け金を 2 倍にし、2 連勝したときに最初のユニットに戻る、というカウンティング手法である。また 8 連敗した場合も最初のユニットに戻る。これらの数字を全て足すと 31 になることから、31 システムと呼ばれる。例えば、1 ユニットを 100 と設定した場合、はじめの賭け金は 100、1 回目の勝負で負けた場合も 100、2 回目の勝負で負けた場合も 100、3 回目に勝利した場合には 200 となる。このカウンティング手法を使用し賭けていき、8 敗したとしても、トータルの損失が 31 ユニットしかないため、資金管理がしやすく、また仕組み的に、2 連勝すればどこのタイミングでも利益が出せるというメリットがある。この手法に似ているカウンティング手法としてマーチンゲール法があるが、そちらに比べて賭け金の増え方が緩やかであることに加え、賭け金の上限に引っかかる可能性も低い。

ココモ法

ココモ法とは、3 回目以降の賭け金を、前回の賭け金 + 前々回の賭け金で導出するカウンティング手法である。1 回目と 2 回目の賭け金は 1 ユニットである。例えば、1 ユニットを 100 と設定した場合、1 回目と 2 回目は 100、3 回目は 100+100 で 200、4 回目は 200+100 で 300、5 回目は 300+200 で 500、6 回目は 500+300 で 800 となる。例からわかる通り、ゲーム回数を重ねるにつ

れて、賭け金は指数関数的に増えてゆき、負けが連続しても一度勝利すればそれまでの損失分もしくはそれ以上の利益を得ることがメリットである。逆に賭け金の上限に引っかかった場合、破綻するというのがデメリットである。この手法に似ているカウンティング手法に、マーチンゲール法があるが、そちらに比べて賭け金の増え方は緩やかである。この手法は、勝率が約 30% であり、勝利した際の払い戻しが 3 倍のゲームで一番利得が出るようになっており、ブラックジャックのような勝率が約 50% であり、勝利した際の払い戻しが 2 倍のゲームではあまり効果がみられないという特徴もある。

（※文責: 菱田美紗紀）

1.5.6 まとめ

ベッティングシステムについて調査した結果、ベッティング手法は、勝ち負けに応じて賭け額を変化させる手法、数列を用意し、その数列を操作しながら賭け額を変化させる手法、資金の何割かを賭ける手法、勝率に応じて賭け額を変化させる手法、その他の手法の 5 種類のグループに分けることができた。そのうち、勝ち負けに応じて賭け額を変化させる手法と数列を用意し、その数列を操作しながら賭け額を変化させる手法、その他の手法は、カウンティング値を参照するのが最初の 1 度のみで、その後はカウンティング値が低くなっても、途中で賭け額を決めることができない。また、資金の何割かを賭ける手法についても、賭け額が資金によって決定されているのでカウンティング値を参照しない。これらの点から、勝率に応じて賭け額を変化させる手法がカウンティングに適用しやすい手法だと考えられる。

（※文責: 柏田輝）

第 2 章　プロジェクトの目標

本プロジェクトでは、次の 3 つの目標を設定した。

目標 1　ブラックジャックの最終的な勝率が 5 割以上になること

- 後述するが、シミュレーションを行った結果、ブラックジャックでベーシックストラテジーを使用した場合の勝率は 4 割程度にしかならなかった。ブラックジャックにおける最終的な利得を増やすためには勝負に勝ち越す必要があると考え、勝率が 5 割以上になる戦略を新たに探索することを目標とした。

目標 2　人間にとって扱いやすい戦略であること

- 勝率が高い戦略を作り出すことができても、実際のカジノではコンピュータを使用することはできない。そしてカジノで実際に戦略を扱うのは人間である。そのため勝率だけではなく、人間にとっての戦略の扱いやすさという点でも優秀である戦略を見つけることとを目標とした。

目標 3　戦略を使用していることがカジノ側に検知されにくいこと

- プレイヤーが戦略を使用していることがカジノ側に検知されてしまった場合、その戦略もまたカジノ側に対策されてしまう可能性がある。そのためカジノ側がどのような基準でプレイヤーが戦略を使用していることを検知しているのかを調べ、対策されないような戦略を作り出すことを目標とした。

以上の 3 点を本プロジェクトの総合的な目標とし、それを達成するため前期と後期でそれぞれ活動目標を立てた。

（※文責: 伊藤晋之介）

第 3 章　複雑性について

3.1　複雑性の定義について

プレイヤーに扱いやすい戦略とは何かについて戦略の複雑性を定義し、複雑性の低い戦略をプレイヤーに扱いやすい戦略であるとした。戦略の複雑性は Chaitin(1969) によって定義されたコルモゴロフ複雑性を参考に新たに定義した。コルモゴロフ複雑性とは、ある文字列があったときに、その文字列を生成するためのプログラムの内、最小の命令長をその文字列の複雑性であると定義したものである。

新たに定義した複雑性を算出するためには、まず戦略の表を文字列として文字列を圧縮する。圧縮の方法としては、元の表を「連続する文字 + 連続して文字が出た回数」と変換することで文字列を圧縮した。例として、「HHSSSHHHHH」という 10 文字からなる文字列を圧縮すると、「H2S3H5」となり、圧縮した後の文字列は 6 文字となる。また「HHHHHHHHHH」という文字列を「H10」と圧縮したときのように、連続して同じ文字が出た回数が 2 桁の場合には数字部分を 1 文字として数える。つまり「H10」の文字列長は 2 文字である。

圧縮した後の文字列長を元の表の大きさ (80) で割った数値を複雑性の定義とした。

（※文責: 米村祥裕）

3.2　複雑性の検証実験

3.2.1　概要

本プロジェクトの目標の要件を満たす戦略表を作成するには、人間が扱いやすいということについて定量的に表し、既存、もしくは新たに作成した戦略表を評価しなければならない。そのため、前期ではコルモゴロフ複雑性を参考にし、連長圧縮を用いることで複雑性を定義し、これをある戦略表を記憶する難易度の評価指標として用いた。しかし、実際に人間がある戦略表を覚える時に感じる難易度と複雑性によって表した難易度は、異なっている可能性がある。そのため、後期では複雑性の検証実験として、いくつかの戦略表を用意し、記憶してもらうという実験を行うことで、複雑性が戦略表を記憶する難易度を正しく表しているかを確かめた。

（※文責: 鳥谷航大）

3.2.2　実験の目的

本実験のもっとも重要な目的は、先述の通り、人間が扱いやすい戦略表というのはどういう戦略表なのかを定量的に表すことである。そのため、本実験では、以下の 2 つを主な目的とした。

Beat the Dealer! Mathematics of Complex Systems and Simulation.

- 目的 1:本プロジェクトが定義した複雑性が戦略表を記憶する難易度を正しく表しているかを確認すること
- 目的 2:目的 1 を達成出来なかった場合、可能な限り戦略表を記憶する難易度を正確に評価できる指標を見つけること

目的 2 について、具体的には、人間のどのような能力がブラックジャックをプレイする能力と関連性を持つかや個人の性格が実験に影響を及ぼすかなどの観点を中心に実験を行った。

（※文責: 鳥谷航大）

3.2.3 仮説

本プロジェクトでは以下の仮説を立て、実験を行った。

- 仮説 1:複雑性は人間が戦略表を記憶する時に感じる難易度を正確に表すことが出来ない
- 仮説 2:一般的な認知的判断能力と戦略表を扱う能力は同じ能力、もしくは深い関連性を持つ
- 仮説 3:リスクを好む・好まない等の性格によって戦略表を記憶し扱う時に違いが生じる

それぞれの仮説に対して詳しく説明する。まず仮説 1 についてであるが、このような仮説を立てた背景として、前期でこの複雑性を用いて性能を評価した時、戦略表の勝率があまり良くない戦略表が高い性能と評価されてしまったことがある。このことから、他に戦略表を記憶する難易度を正確に表すことが出来る指標が存在するのではないか、と考えこのような仮説を立てた。そして、仮説 2、仮説 3 は、先述の他の評価指標として考えられるものである。仮説 2 は、個人の一般的な認知判断能力が評価指標になるのではないかという仮説である。それに対して、仮説 3 は、個人のリスクに関する性格が評価指標になるのではないかという仮説である。

（※文責: 鳥谷航大）

3.2.4 実験準備

今回の実験の目的や仮説に合わせ、実験に用いる戦略表や被験者、報酬などは以下のように準備を行った。まず、今回の実験で用いた戦略表は以下の表 3.1、表 3.2、表 3.3 の 3 つである。

表 3.1　戦略表 A

		ディーラーのアップカード									
		2	3	4	5	6	7	8	9	10	A
手札の合計	5〜8	H	H	H	H	H	H	H	H	H	H
	9	H	H	H	H	H	H	H	H	H	H
	10	H	H	H	H	H	H	H	H	H	H
	11	H	H	H	H	H	H	H	H	H	H
	12	H	H	H	H	H	H	H	H	H	H
	13	H	H	H	H	H	H	H	H	H	H
	14	H	H	H	H	H	H	H	H	H	H
	15	H	H	H	H	H	H	H	H	H	H
	16	S	S	S	S	S	S	S	S	S	S
	17 以上	S	S	S	S	S	S	S	S	S	S

表 3.2　戦略表 B

			ディーラーのアップカード									
			2	3	4	5	6	7	8	9	10	A
手札の合計	ハードハンド	9	H	H	H	H	H	H	H	H	H	H
		10	H	D	D	D	D	H	H	H	H	H
		11	D	D	D	D	D	D	D	D	H	H
		12	D	D	D	D	D	D	D	D	D	D
		13	H	H	S	S	S	H	H	H	H	H
		14	S	S	S	S	S	H	H	H	H	H
		15	S	S	S	S	S	H	H	H	R	H
		16	S	S	S	S	S	H	H	R	R	R
		17 以上	S	S	S	S	S	S	S	S	S	S
	ソフトハンド	A,2	H	H	H	D	D	H	H	H	H	H
		A,3	H	H	H	D	D	H	H	H	H	H
		A,4	H	H	D	D	D	H	H	H	H	H
		A,5	H	H	D	D	D	H	H	H	H	H
		A,6	H	D	D	D	D	H	H	H	H	H
		A,7	S	D	D	D	D	S	S	H	H	H
		A,8	S	S	S	S	S	S	S	S	S	S
		A,9	S	S	S	S	S	S	S	S	S	S
		A,10	S	S	S	S	S	S	S	S	S	S
	スプリット可能	A,A	P	P	P	P	P	P	P	P	P	P
		2,2	P	P	P	P	P	P	H	H	H	H
		3,3	P	P	P	P	P	P	H	H	H	H
		4,4	H	H	H	P	P	H	H	H	H	H
		5,5	D	D	D	D	D	D	D	D	H	H
		6,6	P	P	P	P	P	H	H	H	H	H
		7,7	P	P	P	P	P	P	H	H	H	H
		8,8	P	P	P	P	P	P	P	P	P	P
		9,9	P	P	P	P	P	S	P	P	S	S
		10,10	S	S	S	S	S	S	S	S	S	S

表 3.3　戦略表 C

			ディーラーのアップカード									
			2	3	4	5	6	7	8	9	10	A
手札の合計	ハードハンド	9	H	H	H	H	H	H	H	H	H	H
		10	H	D	D	D	D	H	H	H	H	H
		11	D	D	D	D	D	D	D	D	H	H
		12	D	D	D	D	D	D	D	D	D	D
		13	H	H	S	S	S	H	H	H	H	H
		14	S	S	S	S	S	H	H	H	H	H
		15	S	S	S	S	S	H	H	H	H	H
		16	S	S	S	S	S	H	H	H	H	H
		17 以上	S	S	S	S	S	S	S	S	S	S
	ソフトハンド	A,2	H	H	H	H	D	H	H	H	H	H
		A,3	H	H	H	D	D	H	H	H	H	H
		A,4	H	H	D	D	D	H	H	H	H	H
		A,5	H	H	D	D	D	H	H	H	H	H
		A,6	H	D	D	D	D	H	H	H	H	H
		A,7	D	D	D	D	D	S	S	H	H	H
		A,8	S	S	S	S	D	S	S	S	S	S
		A,9	S	S	S	S	S	S	S	S	S	S
		A,10	S	S	S	S	S	S	S	S	S	S
	スプリット可能	A,A	P	P	P	P	P	P	P	P	P	P
		2,2	H	H	P	P	P	P	H	H	H	H
		3,3	P	P	P	P	P	P	H	H	H	H
		4,4	H	H	H	P	P	H	H	H	H	H
		5,5	D	D	D	D	D	D	D	D	H	H
		6,6	P	P	P	P	P	H	H	H	H	H
		7,7	P	P	P	P	P	P	H	H	H	H
		8,8	P	P	P	P	P	P	P	P	P	P
		9,9	P	P	P	P	P	S	P	P	S	S
		10,10	S	S	S	S	S	S	S	S	S	S

またこれらの戦略表の複雑性は表 3.4 のとおりである。

表 3.4　各戦略表の複雑性

	戦略表 A	戦略表 B	戦略表 B
複雑性	0.2	0.448	0.407

次に被験者についてである。被験者は本学の学生から募集をした。募集方法としては、ブラックジャックに関する実験であることや、戦略表を記憶するテストであるなどの実験の内容を事前に知らせずに募集した。募集の結果、男性 15 人、女性 5 人の計 20 人集めることが出来た。被験者の

年齢は 18〜21 歳であった。

今回 20 人の被験者が集まったため実験時に 2 つのグループに分けて実験を行った。分けられたグループは記憶する戦略表が異なる。具体的な分け方は以下のとおりである。

- グループ X:戦略表 A、B を記憶するグループ
- グループ Y:戦略表 A、C を記憶するグループ

このようにグループ分けした理由としては、比較的複雑性の低い戦略表 A の成績を確認することで、戦略表 B と戦略表 C の成績の違いの要因が、戦略表の複雑性によってなのか個人差によってなのかを確認するためである。

最後に実験協力の報酬についてである。報酬は実験前に、実験結果の成績に応じて配られるという説明を行い、成績が高い被験者ほど多くの報酬が得られるように分配した。

（※文責: 鳥谷航大）

3.2.5 実験の手法・手順

今回の実験は、異なる複雑性を持つ戦略表を 3 つ用意し、先述したグループごとに対応する戦略表を記憶してもらった後テストを行うというものである。その後、CRT(Cognitive Reflection Test) やリスク回避性のテストなどを行い、テストの成績との関連性を分析した。詳しい実験手順は以下のとおりである。

1. ブラックジャックについての説明を行う
2. 5〜10 分程度の時間を用いて実際にプレイしゲームを理解してもらう
3. 戦略表を配り 10 分間で記憶してもらう
4. テストを配り 7 分間で回答してもらう
5. 回答を配り被験者に採点をしてもらいテストを回収する
6. 4〜6 を戦略表を変更しもう一度繰り返す
7. CRT とリスク回避性のテストについて回答してもらう。回答時間は両方合わせて 10〜15 分とした。
8. 結果に応じた報酬を配り実験を終了する

（※文責: 鳥谷航大）

3.2.6 実験結果

今回の実験で得られた主な結果は以下の 3 つである。

- 結果 1:テストの成績と複雑性には強い負の相関があること
- 結果 2:CRT の点数とテストの成績には相関がないこと
- 結果 3:リスク回避性とテストの成績には相関がないこと

ここでは、私達が特に興味深く感じた結果 1,2 に関して詳しく説明していく。

表 3.5 がグループ X の成績であり、表 3.6 はグループ Y の成績である。また、戦略表のテストは満点が 30 点であり、CRT の満点は 11 点である。

表 3.5　グループ X の成績

	戦略表 A	戦略表 B	CRT
平均	29.1	13.8	7.2
分散	2.29	12.96	3.96
標準偏差	1.51	3.6	2.098

表 3.6　グループ Y の成績

	戦略表 A	戦略表 C	CRT
平均	29.8	17.5	8.8
分散	0.6	2.872	1.36
標準偏差	0.36	8.25	1.229

両方のグループともに、戦略表 A は高い成績となった。そのため、成績と CRT 正答率つまり、一般的な認知的判断能力の相関関係を見る時には、戦略表 A から戦略表 B または C の点数を引いた差と CRT 正答率との相関を考えた。図 3.1 がグループ X の散布図であり、図 3.2 がグループ Y の散布図である。

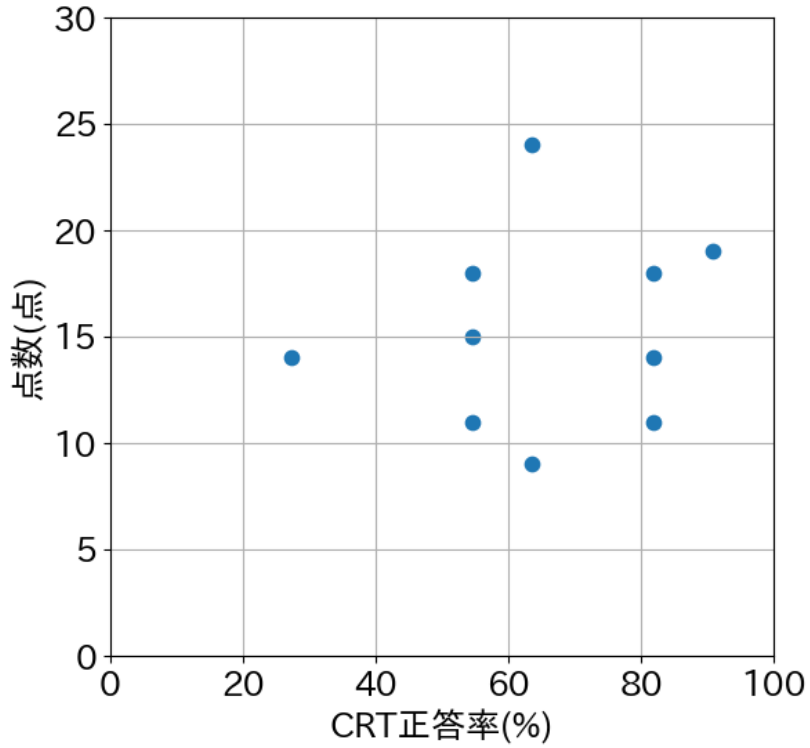


図 3.1　CRT 正答率と戦略表 A から戦略表 B の点数を引いた値との散布図

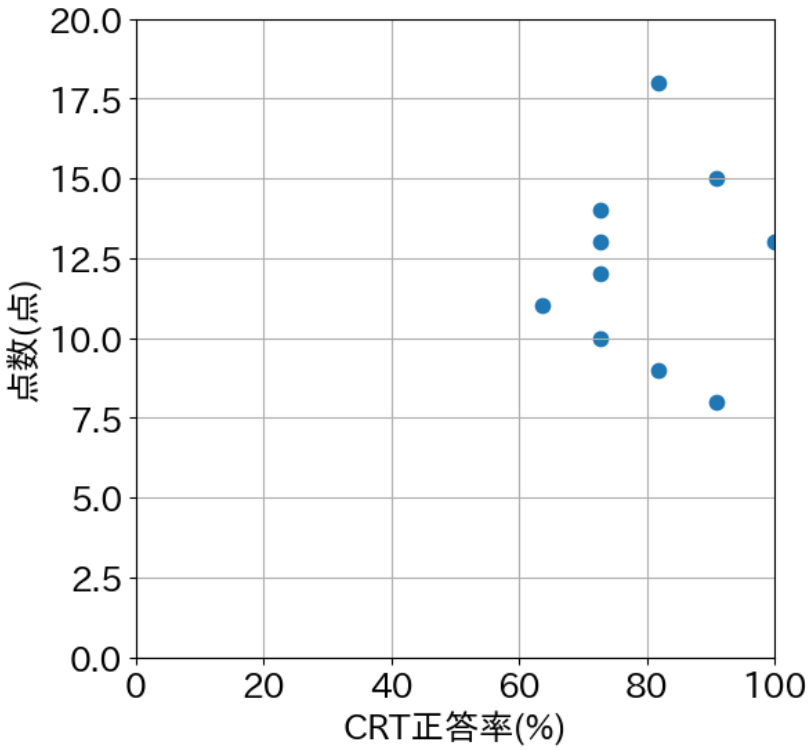


図 3.2　CRT 正答率と戦略表 A から戦略表 C の点数を引いた値との散布図

また、この散布図より相関係数を求めたところ表 3.7 のようになった。

表 3.7　それぞれのグループの相関係数

	グループ X	グループ Y
CRT と点数の差の相関係数	0.145	0.079

上の表の通り、相関係数は非常に小さいという結果になった。ただし、相関係数が小さくても相関がある可能性があるため、無相関検定を行った。無相関検定の検定条件などは後述の複雑性と成績の相関係数と合わせて詳しく後で述べる。無相関検定の結果、グループ X、グループ Y ともに相関が無いことがわかった。

次に、複雑性とテストの成績の関連性についてである。以下の表 3.8 が各戦略表の複雑性とテストの平均点である。

表 3.8　各戦略表の複雑性とテストの平均点

	戦略表 A(グループ X)	戦略表 A(グループ Y)	戦略表 B	戦略表 C
テストの平均点	29.1	29.8	13.8	17.5
複雑性	0.2	0.2	0.448	0.407

さらに、実験で得られたデータを用いて成績と複雑性の相関係数を求めたところ -0.942 となり、ほぼ -1 に近い値となった。こちらも無相関検定を行ったところ、非常に強い負の相関があるという結果になった。

（※文責: 鳥谷航大）

3.2.7　実験結果の検定と分析

ここでは、先程の無相関検定についての説明と各戦略表のテストの平均点に差があるかどうかを調べるために行った分散分析と多重比較について述べる。

まずは無相関検定についてである。無相関検定は、標本から得た相関係数の値が母集団でも同様に成立するかどうかを調べるために行う。標本の大きさを n、標本相関係数を r_{xy} とし、統計量を t_0 とすると、次式のようになる。

$$t_0 = \frac{|r_{xy}|\sqrt{n-2}}{\sqrt{1-r_{xy}^2}} \tag{3.1}$$

以下がグループ X・グループ Y の成績とそれぞれの CRT の正答率との相関を調べる無相関検定の検定条件の表である。

表 3.9　無相関検定の検定条件

	グループ X	グループ Y
帰無仮説	相関係数が 0	
対立仮説	相関係数が 0 ではない	
相関係数	0.145	0.079
自由度	8	
T 値	0.415	0.224
有意水準	0.05	
P 値	0.689	0.829

以上より、先述の通り、グループ X・グループ Y ともに相関係数は 0 となり、CRT の正答率とテストの成績には相関がないことが確認された。

次に、複雑性とテストの成績に関する無相関検定である。同様に、以下が検定条件の表である。

表 3.10　無相関検定の検定条件

	複雑性と成績
帰無仮説	相関係数が 0
対立仮説	相関係数が 0 ではない
相関係数	-0.942
自由度	18
T 値	7.963
有意水準	0.05
P 値	0.000

以上より、こちらも先述の通り、複雑性とテストの成績には強い負の相関があることが確認された。

次に、分散分析と多重比較についてである。今回の実験で得られたデータの性質から一元配置分散分析という手法を使用した。戦略表 A、B、C のテストの平均点に差があるかどうかを検定した。そして平均点に有意差があった場合には多重比較を用いて、どの戦略間で有意差があるかを検証する。今回の実験では、戦略表 A の複雑性が 0.2 となっていたためか、他のテストの成績に比べとても高い成績となった。そのため、先述の相関係数が負の相関を表したことが戦略表 A の結果に強く影響を受けている可能性や、B,C 間の成績に違いがない可能性があるため分散分析と多重比較を行った。以下が、その検定条件と結果である。

表 3.11　分散分析と多重比較の検定条件

	全ての戦略表間	A-B 間	A-C 間	B-C 間
帰無仮説	各戦略表の平均点に差がない	戦略表間に差がない		
対立仮説	各戦略間に差がある	戦略間に差がある		
有意水準	0.05			
P 値	0.000	0.000	0.000	0.007

以上より、全ての戦略表の成績に差があり、かつ全ての戦略表間の成績に差があるという結果になった。

（※文責: 薩田凱斗）

3.2.8　考察

以上の結果を全てまとめると、以下のようになる。

- 結果 1:複雑性とテストの成績には強い負の相関がある
- 結果 2:CRT の正答率とテストの成績には相関がない
- 結果 3:リスク回避性とテストの成績には相関がない

以上の実験結果より、以下のようにまとめることが出来る。

- 考察 1:戦略表の複雑性が高いほどテストの成績は悪くなるため、複雑性は評価指標として適している
- 考察 2:ブラックジャックの戦略を用いる能力は一般的な認知的判断能力とは異なる
- 考察 3:個人のリスクを好むかどうかなどの正確はブラックジャックをプレイする上で大きな影響は与えない

以上より、本プロジェクトで定義した複雑性が適していることが確認されたため、実験は概ね成功だった。しかし、ブラックジャックをプレイするのに必要な能力が明確になっていないため、今後の実験で検証していきたい。

（※文責: 鳥谷航大）

3.2.9　実験の課題点

実験の結果としては最終的に成功したと言える。しかし実際には、実験のプロセスでいくつかの問題点があった。そこで、今後の実験をする上での課題点としてここにまとめておく。実験を行う前の準備段階では、ブラックジャックの戦略表を暗記する実験のテストと CRT に加え、マジカルナンバーという短期記憶力を測定するテストも行う予定であった。マジカルナンバーとはアメリカの認知心理学者である Miller(1956) による論文のタイトルからそう呼ばれているものであり、マ

ジカルナンバーは認知心理学において 7 ± 2 という数字である。Miller の実験によると、人間が瞬間的に記憶できる情報の最大数は 5 から 9、つまり 7 ± 2 の範囲であることがわかっている。そこで、ブラックジャックの戦略表を暗記する能力は短期記憶力が高い人ほど優れているという仮説を立てて、マジカルナンバーの実験も行う必要があった。今回の実験で行う予定だったマジカルナンバーのテストは、スライドに 9 桁のランダム生成された数字を 5 秒間表示し、それを被験者に記憶してもらい回答してもらうという流れだった。それをしかし準備不足や実験時間の調整ミスなどにより実際には 20 人中、7 人の被験者にしか行うことができなかった。データ数が少なくその仮説を検証するまでに至らなかったため、戦略表の記憶力と短期記憶との相関を調べることができなかった。また、実験で使用した記憶してもらう 3 種類の戦略は全て色をつけて表にした。なので、被験者は表を色の分かれ目などを絵として写真記憶のような能力で記憶することができてしまう。これでは戦略の文字列を記憶する単純な記憶能力とは異なるため本来の実験で測る記憶能力の意図とは外れてしまう。なので、今回の実験で使用した文字列を圧縮して算出した複雑性を、戦略表を画像としてその圧縮率を複雑性として用いる必要があった。また、今回の実験の報酬に関してであるが、上記した通り成績が良ければ良いほど、より多くの報酬をもらうという形だった。しかし、ブラックジャックのゲームの賭け金の流れと同様の条件として、ゲームに負けると賭け金を没収されるように実験の報酬も戦略表の問題を間違えると報酬が減るという条件をつけると、よりブラックジャックのゲームに近い状況になるのではないかと考えた。そのため、今後の実験では問題を間違えて回答した数に応じて報酬を減らしていくという形をとりたい。

（※文責: 薩田凱斗）

第 4 章　シミュレータの作成について

4.1　概要

今回ブラックジャックの戦略を検証するためにデック数を無限に設定したものとデック数を 1 に設定したものとで 2 種類の条件を用意しシミュレーションを実行した。その際に勝率、デック数、複雑性のそれぞれの観点から仮説を設定した。また、シミュレーションを実行するためのシミュレータを Python3 を用いて作成した。その際に Python Software Foundation(2018) のドキュメントを参照した。以下の部分ではシミュレーションの条件設定、デック数についての説明とデック数の違いが生む影響、設定した仮説、シミュレータの詳細のそれぞれについて順に述べていく。

（※文責: 尾崎拓海）

4.2　シミュレーションの条件設定

今回 7 つの戦略で実験を行う。その際、2 つの条件を設定した。

- 条件 1:デック数が無限であるとする
- 条件 2:デック数が 1 つであるとする

条件 1 は 13 種類のカードすべてを常に同じ確率で引くという状況に相当する。カードを何十枚、何百枚引いたとしても次に引くカードの確率は一定であり、場に出たカードに影響を受けないという状況である。
条件 2 は次に引くカードが場にあるカードに影響を受けるという状況に相当する。次に引くカードの確率は既にゲームに使用されたカードや場に出ているカードによって変動するという状況である。
2 つの条件で比較する際に共通している条件は以下の通りである。

- プレイヤーの人数とディーラーの人数はそれぞれ一人とする。
- ベーシックストラテジー、ベーシックストラテジー改変 1、ベーシックストラテジー改変 2、15 以上になるまでヒットする戦略、16 以上になるまでヒットする戦略、17 以上になるまでヒットする戦略、18 以上になるまでヒットする戦略の、それぞれの戦略でシミュレータを実行する。
- ゲームの実行回数は 10 万回とする。

（※文責: 尾崎拓海）

4.2.1　デック数について

今回デックはジョーカーを抜いた 52 枚 1 組を 1 デックと表す。例えば、8 デックを使用するとした場合、1 デック 52 枚である事から、$52 \times 8 = 416$、合計 416 枚のカードがゲームに使用される。また、一般的にはブラックジャックで使用されるデック数は 1,2,4,6,8 デックのいずれかである場合が多い。
デック数によってデックの枚数や含まれるそれぞれのカードの枚数などが変化し、それがカードを引く確率に影響を与える。そのためブラックジャックの勝敗にも影響を及ぼす。

（※文責: 尾崎拓海）

4.2.2　デック数の違いによる影響

今回のシミュレーションで比較するデック数無限とデック数 1 の双方において、10 を 1 枚引いた後で更に続けて 10 を引く確率を考えてみる。

デック数無限の場合

デック数が無限である事からデックに含まれる 10 の枚数も同様に無限となっている。この状態のデックから 10 を 1 枚引いたとしてもデックに残っている 10 の枚数は変わらず無限であり、次に 10 を引く確率は一定となる。この事から次にデックから 10 を引く確率は

$$\frac{4}{52} \fallingdotseq 7.69\ \% \tag{4.1}$$

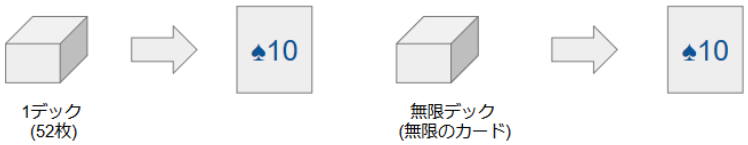
となる。

デック数 1 の場合

デック数 1 の場合使用するカードは 52 枚でありその中に 10 は 4 枚含まれている。この状態のデックから 10 を 1 枚引くとデックに含まれる 10 の枚数は 1 枚少なくなる。この事から次にデックから 10 を引く確率は

$$\frac{3}{51} \fallingdotseq 5.88\ \% \tag{4.2}$$

となる。
デック数無限とデック数 1 の条件とで 10 を引いた後に続けて 10 を引く確率を比較してみるとデック数無限のときはおよそ 7.69 %、デック数 1 の時はおよそ 5.88 %となり、デック数により次に引くカードの確率が変化していることが分かる。これがデック数の違いによる影響である。以下の図 4.1 はデック数による勝率の違いを表したものである。



1デック
(52枚)

無限デック
(無限のカード)

次に10を引く確率：3/51
≒5.88%

次に10を引く確率：4/52
≒7.69%

図 4.1　デック数による勝率の違い

（※文責: 尾崎拓海）

4.3　ブラックジャックシミュレータ

ブラックジャックのシミュレータをプログラミング言語（Python3）を用いて作成した。このシミュレータを使用して、ベーシックストラテジー、ベーシックストラテジー改変 1、ベーシックストラテジー改変 2、15 以上になるまでヒットする戦略、16 以上になるまでヒットする戦略、17 以上になるまでヒットする戦略、18 以上になるまでヒットする戦略のそれぞれについて勝ち、負け、引き分けの 3 つを調べた。また、ブラックジャックを行った際の資金の推移や勝率の推移といったものを調査するためにシミュレータに所持金の概念を導入したり、実際の人間のように戦略を間違えるという処理を追加し、ダブルダウンやスプリット、サレンダーといったプレイヤー側の選択肢を追加した。このシミュレータを使用して様々な実験を実施した。ここではシミュレータ内部の詳細について記述していく。

（※文責: 尾崎拓海）

基本設計

まず初めに、シミュレータの基本設計について説明する。今回作成したシミュレータではブラックジャックを行う際に必要となる要素をクラスとして表現した。具体的にはトランプのカードを表現するカードクラスとそれを一纏めにするデッククラス、ゲーム参加者を表すクラスとそれを継承したプレイヤークラスとディーラークラス、ゲームの勝敗を判定するマネージャークラスのそれぞれを定義した。これらのクラスを用いてブラックジャックのゲームを再現し、ベーシックストラテジーとその他の戦略を実行する。次に各クラスの詳細を記述していく。

トランプのカードを表現するクラス

このクラスでは実際のトランプのカードを表現するために rank という変数に A〜K というトランプのランクを、suit という変数にスペード、ハート、ダイヤ、クラブのスートを定義した。また、J,Q,K,A の絵札カードは 10 や 11 と数える必要があったので、ランクを数字に変換する処理もこのクラスに書き、value という変数に入力した。

デックを表現するクラス

このクラスでは先程定義したカードクラスを利用してデックを定義した。具体的には先程のカードクラスの配列を作成し、その中にジョーカーを除く 52 種類のトランプカードを作成した。このクラスの初期化時に使用するデックの数を指定する。また、デックのシャッフルには独自に作成した関数を使用した。このシャッフル関数は Python3 の random 関数を用いて独自に設計したものであり、引数にシャッフルを行う回数を指定する。カードの配列の長さが仮に 52 だった場合には、1〜26 番目のカードからランダムに取り出したカードと、27〜52 番目のからランダムに取り出したカードを交換するという処理を（デック数×指定されたシャッフル回数）繰り返すという処理でシャッフル関数を作成した。

ゲーム参加者を表すスーパークラス

このクラスでは自身の手札とその手札の合計値、手札に含まれる A の枚数、バーストしているかどうかのフラグ、手札がブラックジャックとなっているかどうかのフラグのそれぞれを定義している。手札に含まれる A の枚数は自身の手札の合計値を計算する時と、ブラックジャックの条件を満たしているかどうかを判別する際に使用した。また手札の合計値を返す関数を定義し、その内側で自身がバーストしているかどうかの判定も行っている。

プレイヤークラス

このクラスは先のゲーム参加者を表すスーパークラスを継承しており、ゲームに参加しているプレイヤーを表現している。プレイヤークラスでは新たに自身の名前を表す変数と自身の勝利回数、敗北回数を記録する変数を定義した。またこのクラスでは新しく、カードを受け取る関数とヒットを行う関数、スタンドを行う関数、勝利回数と敗北回数を増加させる関数を作成した。また、ヒット、スタンド、ダブルダウン、サレンダーの処理を行う関数を作成した。

ディーラークラス

このクラスは先のゲーム参加者を表すスーパークラスを継承しており、ゲームのディーラーを表現しているクラスとなっている。ディーラークラスの中でデックをインスタンス化してディーラー側がデックを所持している事を表現している。このクラスでは新しく、デックのシャッフル回数という変数を定義した。また、このクラスではカードを配る関数、ディーラーの手札合計が 17 を超えるまでカードを引き続ける関数を作成した。カードを配る関数についてはデック数有限の時とデック数無限の時とで処理を変更している。また、カウンティングを行う際に使用されたカードを数える必要があったので、使用されたカードに対応してカウントを増減させる処理をカウンティング手法ごとに記述した。

ゲームマネージャークラス

このクラスは主にゲームの勝敗判定に使用している。プレイヤーとディーラーの手札の合計値を比較し勝敗を判定する関数と、手札がブラックジャックになっているかどうかを判定する関数を作

成した。また、プレイヤーの勝敗に応じて資金を移動させる処理を記述した。勝敗判定のタイミングでプレイヤーの勝利回数、敗北回数のそれぞれを記録している。

メイン関数

以上のクラスを用いてメイン関数にブラックジャックのゲームを記述した。以下にプログラムの実行手順を示す。

1. ゲームに参加するプレイヤーを作成。今回はプレイヤーを一人のみ作成した。
2. ディーラーを作成。
3. カットカードを定義。カットカードを挟む位置はデックの半分の位置とした。
4. ゲーム全体の実行回数を定義。今回は 10 万回とした。
5. プレイヤーの戦略を配列形式で定義した。
6. ゲームを繰り返す while 文を作成し、ループ回数を 10 万回とした。
   （a）デックからカットカードが出てきたかを確認する。もし出てきていればデックをシャッフルする。
   （b）ディーラーが自身を含む各プレイヤーに初期カードを配る。
   （c）プレイヤーは自身の戦略に応じて掛け金を決定する。
   （d）プレイヤーは自身の戦略に沿った行動を選択する。
   （e）プレイヤーは戦略ごとに一定の確率で間違えた行動を選択する。
   （f）すべてのプレイヤーの行動が終了したことを確認後にディーラーが行動を開始する。
   （g）ディーラーの行動終了後に、勝敗判定を行う。

エラー時の行動

戦略の複雑性に応じてエラーのしやすさを定義した。エラーした際の行動は以下のようになる。

- ヒットとスタンドのみで構成された戦略の場合
  - ヒットでエラー：スタンド
  - スタンドでエラー：ヒット
- ヒット、スタンド、ダブルダウン、スプリットで構成された戦略の場合
  - ヒットでエラー：80 %スタンド、20 %ダブルダウン
  - スタンドでエラー：80 %ヒット、20 %ダブルダウン
  - ダブルダウンでエラー：50 %ヒット、50 %スタンド
  - スプリットでエラー：スプリットを行わない

（※文責: 尾崎拓海）

4.4　シミュレータの擬似乱数の検証

今回シミュレータを作成するにあたり、擬似乱数を使用した。この擬似乱数が適切かどうかについて検証する。今回使用した擬似乱数生成方法は Python3 の random 関数である。Python Software Foundation(2018) によれば random の擬似乱数を生成するアルゴリズムはメルセンヌツイスタを用いている。今回は周期と広井 (2007) の等確率性の検定を行う。

（※文責: 柿崎大輝）

4.4.1　周期

擬似乱数には周期が存在する。周期とは同じ数列が出てくるようになるまでの数字の出現回数のことを指す。周期が小さいとよく同じ数列が出てきてしまいランダム性が低い。つまり周期が大きいとランダム性が高いので、性能が良いということになる。松本 (2013) ではメルセンヌツイスタの周期は $2^{19937}-1$ である。これはほかの擬似乱数に比べ、かなり大きい周期である。そのため、メルセンヌツイスタを擬似乱数として使うのに十分であると考えられる。

（※文責: 柿崎大輝）

4.4.2　等確率性の検定

等確率性とはどの値も等しい確率で出てくるかどうかである。カイ 2 乗検定を使い等確率性を検証する。random 関数を使用して、0〜1 の範囲の乱数を生成する。乱数の生成後、生成された値を 0〜0.1,0.1〜0.2,0.2〜0.3,0.3〜0.4,0.4〜0.5,0.5〜0.6,0.6〜0.7,0.7〜0.8,0.8〜0.9,0.9〜1.0 の 10 通りに分類する。それをまとめると表 4.1 になる。

表 4.1　random 関数での結果

区分	0〜0.1	0.1〜0.2	0.2〜0.3	0.3〜0.4	0.4〜0.5	0.5〜0.6	0.6〜0.7	0.7〜0.8	0.8〜0.9	0.9〜1.0
度数	95	85	100	102	91	114	87	108	115	103

完璧なランダムなのであれば、この結果はどれも 100 になることが予想できる。しかし、実際はすべてが 100 にはならないので、カイ 2 乗検定を行い検証する。表 4.1 に示した度数を実現度数として使用し、100 を理論度数としてカイ 2 乗検定を行う。この時、自由度は 9 で有意水準を 5 %とすると、棄却値は 16.92 となり、カイ 2 乗値がこれより小さいと擬似乱数が等しく出てきたといえる。実際に計算すると、カイ 2 乗値は 9.98 となった。この値は 16.92 より小さいので、擬似乱数によって出た値はすべて等しい確率で出てきたといえる。以上のことから、メルセンヌツイスタは周期が大きいこととカイ 2 乗検定で値が全て等しい確率で出ていることから十分に使えると判断することができるため、今回のシミュレータにおいて使用した。

Beat the Dealer! Mathematics of Complex Systems and Simulation.

（※文責: 柿崎大輝）

第 5 章　ベーシックストラテジーの検証

5.1　戦略同士の比較

Thorp(1962) によって提案されたベーシックストラテジーについて検証を行う。ベーシックストラテジーの性能評価のために比較対象として 6 つの戦略を考えた。比較対象となる戦略は次の通りである。

- ベーシックストラテジー改変 1
- ベーシックストラテジー改変 2
- プレイヤーの合計値が 15 以上になるまでヒットする戦略
- プレイヤーの合計値が 16 以上になるまでヒットする戦略
- プレイヤーの合計値が 17 以上になるまでヒットする戦略
- プレイヤーの合計値が 18 以上になるまでヒットする戦略

以上の 6 つの戦略について、これから詳細に説明する。

（※文責: 米村祥裕）

5.1.1　ベーシックストラテジー改変 1

ベーシックストラテジーはブラックジャックにおける有効な戦略の一つである。しかし戦略の表に注目すると、表の複雑性を考えたときに変更の余地があると考えた。戦略表においてプレイヤーの合計値が 12 の行に注目する。ディーラーのアップカードが 2,3 の時は H となっているが、その後アップカードが 4,5,6 の時は S、アップカードが 7,8,9,10,A の時は H となっており、H に挟まれて S が存在している。表を覚えることを考えると、同じ行の中で変化が少ない方が複雑性が低いといえる。そのため、ベーシックストラテジー改変 1 では表 5.1 に示すようにプレイヤーの合計値が 12、ディーラーのアップカードが 2,3 の時の戦略を S に変更した。

（※文責: 米村祥裕）

5.1.2　ベーシックストラテジー改変 2

ベーシックストラテジー改変 1 と同様にベーシックストラテジーの戦略表を改変した。変更点はベーシックストラテジー改変 1 と同様にプレイヤーの合計値が 12 の行である。改変 1 ではディーラーのアップカードが 2,3 の部分を S に変更したが、改変 2 では 4,5,6 を H に変更した。この変更によってプレイヤーの合計値が 12 の行はすべて H という表 5.2 ができた。

（※文責: 米村祥裕）

表 5.1　ベーシックストラテジー改変 1

		ディーラーのアップカード									
		2	3	4	5	6	7	8	9	10	A
手札の合計	19 以上	S	S	S	S	S	S	S	S	S	S
	18	S	S	S	S	S	S	S	S	S	S
	17	S	S	S	S	S	S	S	S	S	S
	16	S	S	S	S	S	H	H	H	H	H
	15	S	S	S	S	S	H	H	H	H	H
	13〜14	S	S	S	S	S	H	H	H	H	H
	12	S	S	S	S	S	H	H	H	H	H
	11 以下	H	H	H	H	H	H	H	H	H	H

表 5.2　ベーシックストラテジー改変 2

		ディーラーのアップカード									
		2	3	4	5	6	7	8	9	10	A
手札の合計	19 以上	S	S	S	S	S	S	S	S	S	S
	18	S	S	S	S	S	S	S	S	S	S
	17	S	S	S	S	S	S	S	S	S	S
	16	S	S	S	S	S	H	H	H	H	H
	15	S	S	S	S	S	H	H	H	H	H
	13〜14	S	S	S	S	S	H	H	H	H	H
	12	H	H	H	H	H	H	H	H	H	H
	11 以下	H	H	H	H	H	H	H	H	H	H

5.1.3　プレイヤーの合計値が一定以上になるまでヒットする戦略

ディーラーがある程度有利な戦略を採用しているという仮定の下で、プレイヤーもディーラーと同じような行動を行う戦略を考えた。プレイヤーの手札の合計値が一定以上になるまでヒットするものとして、次の 4 つの戦略を用意した。

- プレイヤーの合計値が 15 以上になるまでヒットする戦略
- プレイヤーの合計値が 16 以上になるまでヒットする戦略
- プレイヤーの合計値が 17 以上になるまでヒットする戦略
- プレイヤーの合計値が 18 以上になるまでヒットする戦略

（※文責: 米村祥裕）

表 5.3　プレイヤーの合計値が 15 以上になるまでヒットする戦略

		ディーラーのアップカード									
		2	3	4	5	6	7	8	9	10	A
手札の合計	19 以上	S	S	S	S	S	S	S	S	S	S
	18	S	S	S	S	S	S	S	S	S	S
	17	S	S	S	S	S	S	S	S	S	S
	16	S	S	S	S	S	S	S	S	S	S
	15	S	S	S	S	S	S	S	S	S	S
	13〜14	H	H	H	H	H	H	H	H	H	H
	12	H	H	H	H	H	H	H	H	H	H
	11 以下	H	H	H	H	H	H	H	H	H	H

表 5.4　プレイヤーの合計値が 16 以上になるまでヒットする戦略

		ディーラーのアップカード									
		2	3	4	5	6	7	8	9	10	A
手札の合計	19 以上	S	S	S	S	S	S	S	S	S	S
	18	S	S	S	S	S	S	S	S	S	S
	17	S	S	S	S	S	S	S	S	S	S
	16	S	S	S	S	S	S	S	S	S	S
	15	H	H	H	H	H	H	H	H	H	H
	13〜14	H	H	H	H	H	H	H	H	H	H
	12	H	H	H	H	H	H	H	H	H	H
	11 以下	H	H	H	H	H	H	H	H	H	H

表 5.5　プレイヤーの合計値が 17 以上になるまでヒットする戦略

		ディーラーのアップカード									
		2	3	4	5	6	7	8	9	10	A
手札の合計	19 以上	S	S	S	S	S	S	S	S	S	S
	18	S	S	S	S	S	S	S	S	S	S
	17	S	S	S	S	S	S	S	S	S	S
	16	H	H	H	H	H	H	H	H	H	H
	15	H	H	H	H	H	H	H	H	H	H
	13〜14	H	H	H	H	H	H	H	H	H	H
	12	H	H	H	H	H	H	H	H	H	H
	11 以下	H	H	H	H	H	H	H	H	H	H

表 5.6　プレイヤーの合計値が 18 以上になるまでヒットする戦略

		ディーラーのアップカード									
		2	3	4	5	6	7	8	9	10	A
手札の合計	19 以上	S	S	S	S	S	S	S	S	S	S
	18	S	S	S	S	S	S	S	S	S	S
	17	H	H	H	H	H	H	H	H	H	H
	16	H	H	H	H	H	H	H	H	H	H
	15	H	H	H	H	H	H	H	H	H	H
	13〜14	H	H	H	H	H	H	H	H	H	H
	12	H	H	H	H	H	H	H	H	H	H
	11 以下	H	H	H	H	H	H	H	H	H	H

5.1.4　各戦略の複雑性

今回用意した戦略を、全て圧縮したのが表 5.7 である。

表 5.7　各戦略の圧縮した後の文字列と複雑性

戦略	圧縮した後の文字列	文字列長	複雑性
ベーシックストラテジー	S10S10S10S5H5S5H5S5H5H2S3H5H10	26	0.325
ベーシックストラテジー改変 1	S10S10S10S5H5S5H5S5H5S5H5H10	24	0.300
ベーシックストラテジー改変 2	S10S10S10S5H5S5H5S5H5H10H10	22	0.275
15 以上になるまでヒットする戦略	S10S10S10S10S10H10H10H10	16	0.2
16 以上になるまでヒットする戦略	S10S10S10S10H10H10H10H10	16	0.2
17 以上になるまでヒットする戦略	S10S10S10H10H10H10H10H10	16	0.2
18 以上になるまでヒットする戦略	S10S10H10H10H10H10H10H10	16	0.2

表 5.7 を見ると、ベーシックストラテジーを改変した戦略の方が、複雑性が低く、プレイヤーにとって扱いやすいといえる。また、手札の合計値が一定以上になるまでヒットする戦略は、複雑性がベーシックストラテジーの半分程度であり、プレイヤーにとって扱いやすい戦略だといえる。

（※文責: 渡邊凛）

5.2　仮説

今回は仮説を以下のように設定した。

- 勝率に関して
  - 仮説 1. デック数が無限の時にはベーシックストラテジーの方が勝率が高い
  - 仮説 2. デック数が 1 の時にはベーシックストラテジー以外の勝率が高い

Beat the Dealer! Mathematics of Complex Systems and Simulation.

- 複雑性を考慮した場合
  - 仮説 3. プレイヤーの合計値が 15,16,17,18 以上になるまでヒットする戦略の方が性能が高い
- デック数を考慮した場合
  - 仮説 4. デック数 1 とデック数無限では勝率に有意な差が出る

ベーシックストラテジーの表はデック数が無限であることを前提として導出されている。我々はこの点に着目し、デック数が有限になった際にはベーシックストラテジーよりも優れた戦略が存在するのではないか、あるいは、ベーシックストラテジーはデック数有限には対応しきれないのではないかと考えた。こうした考えから仮説 1、仮説 2 のそれぞれを設定した。また、基準値以上になるまでヒットする戦略の方が複雑性が低くなり、性能の評価がよくなるのではないかという考えから仮説 3 を設定した。デック数 1 とデック数無限では, カードを引く確率が変化する事から、デック数が違えば勝率に有意な差が出るのではないかと考え、仮説 4 を設定した。

（※文責: 尾崎拓海）

5.3 検証手順

設定した仮説を以下の手順で検証した。

1. ブラックジャックのシミュレータを作成
2. デック数が 1 の場合と無限の場合でシミュレーションを 10 万回実施
3. 勝った割合、負けた割合、引き分けた割合の 3 つを調べた
4. 得られた結果から基本戦略とその他の戦略との間の勝率に有意な差があるかどうかをカイ二乗検定を用いて調べた

（※文責: 尾崎拓海）

5.4 シミュレーション結果

シミュレータを用いて、10 万回ブラックジャックを行った結果を表 5.8 と表 5.9 に示す。

表 5.8　デック数と各戦略での勝利、負け、引き分け

	デック数無限			デック数 1		
	勝ち	負け	引き分け	勝ち	負け	引き分け
ベーシックストラテジー	42746	48635	8619	43111	48654	8235
ベーシックストラテジー改変 1	42583	48782	8635	42923	48909	8168
ベーシックストラテジー改変 2	42223	49079	8698	42955	48689	8356
15 以上になるまでヒットする戦略	42392	49458	8150	42063	50027	7910
16 以上になるまでヒットする戦略	41410	49506	9084	41580	49672	8748
17 以上になるまでヒットする戦略	40870	49400	9730	40974	49630	9396
18 以上になるまでヒットする戦略	39291	52587	8122	42071	49872	8057

表 5.9　デック数と各戦略での勝利、負け、引き分けの比率

	デック数無限			デック数 1		
	勝ち (%)	負け (%)	引き分け (%)	勝ち (%)	負け (%)	引き分け (%)
ベーシックストラテジー	42.7	48.6	8.6	43.1	48.7	8.2
ベーシックストラテジー改変 1	42.6	48.8	8.6	42.9	48.9	8.2
ベーシックストラテジー改変 2	42.2	49.1	8.7	43.0	48.7	8.4
15 以上になるまでヒットする戦略	42.4	49.5	8.2	42.1	50.0	7.9
16 以上になるまでヒットする戦略	41.4	49.5	9.1	41.6	49.7	8.7
17 以上になるまでヒットする戦略	40.9	49.4	9.7	41.0	49.6	9.4
18 以上になるまでヒットする戦略	39.3	52.6	8.1	42.1	49.9	8.1

今回はシミュレータで各戦略を 10 万回実行した。その結果を勝ち、負け、引き分けの 3 種類に分けて、表 5.8 と表 5.9 にまとめた。表 5.8 は度数で表し、表 5.9 では比率で表したものになっている。縦軸はデックの数とそれに対応する勝敗とし、横軸は戦略ごとに分けている。例えば、表 5.9 ベーシックストラテジーでデック数 1 の場合を見ると勝ちが 43.1% で負けが 48.7% で引き分けが 8.2% となっている。

全体として勝ちの数は 4 万回前後に収まっており、負けの数は 5 万回前後に収まっている。基本的に負けの数のほうが多いことが分かる。デック数で比べると、ほとんどの戦略でデック数 1 の場合のほうが勝ちの数が多いということが発見される。また、デック数に関係なく、勝ちの数はベーシックストラテジーが一番多い事が分かる。

（※文責: 薩田凱斗）

5.5　検定

先ほどのシミュレーションを行った結果から、ベーシックストラテジーが各戦略の中で最も勝率が高いという事を明らかにする。そのために、各戦略の間の勝率にそれぞれ有意な差があるかどう

かを確かめたい。また、デック数によって勝率が変化するということも明らかにする。そのためデック数の違いによって各戦略の勝率にそれぞれ有意な差があるかどうかを確かめるために検定を行う。そこで、それらの推測が実際に正しいかどうかを確かめるために、カイ 2 乗検定を用いて検証を行った。

（※文責: 薩田凱斗）

5.5.1 カイ 2 乗検定による独立性の検定

勝率に有意な差があるかどうかを確かめるためカイ 2 乗検定の独立性の検定を使用する。カイ 2 乗検定ではカイ 2 乗値を用いて検定を行う。カイ 2 乗値は式 5.1 で算出する。

$$\text{カイ 2 乗値} = \sum \frac{(\text{実現度数} - \text{理論度数})^2}{\text{理論度数}} \tag{5.1}$$

実現度数とは実際に出た度数のことで、理論度数は理想としてでる度数のことである。実現度数はシミュレータの結果から参照し、理論度数はシミュレータの結果から算出する。理論度数は式 5.2 で計算される。

$$\text{理論度数} = \text{行の合計} \times \frac{\text{列の合計}}{\text{すべての合計}} \tag{5.2}$$

（※文責: 柿崎大輝）

戦略間の勝率

カイ 2 乗検定を行うためにシミュレータの結果の表 5.10 を変形する。

表 5.10　デック数無限の場合

	勝ち	勝ち以外 (負けと引き分け)	合計
ベーシックストラテジー (実現度数)	42746	57254	100000
ベーシックストラテジー (理論度数)	41645	58355	
ベーシックストラテジー改変 1 (実現度数)	42583	57417	100000
ベーシックストラテジー改変 1 (理論度数)	41645	58355	
ベーシックストラテジー改変 2 (実現度数)	42223	57777	100000
ベーシックストラテジー改変 2 (理論度数)	41645	58355	
15 以上になるまでヒットする戦略 (実現度数)	42392	57608	100000
15 以上になるまでヒットする戦略 (理論度数)	41645	58355	
16 以上になるまでヒットする戦略 (実現度数)	41410	58590	100000
16 以上になるまでヒットする戦略 (理論度数)	41645	58355	
17 以上になるまでヒットする戦略 (実現度数)	40870	59130	100000
17 以上になるまでヒットする戦略 (理論度数)	41645	58355	
18 以上になるまでヒットする戦略 (実現度数)	39291	60709	100000
18 以上になるまでヒットする戦略 (理論度数)	41645	58355	
合計	291515	408485	700000

表 5.11　デック数 1 の場合

	勝ち	勝ち以外 (負けと引き分け)	合計
ベーシックストラテジー (実現度数)	43111	56889	100000
ベーシックストラテジー (理論度数)	42240	57760	
ベーシックストラテジー改変 1 (実現度数)	42923	57077	100000
ベーシックストラテジー改変 1 (理論度数)	42240	57760	
ベーシックストラテジー改変 2 (実現度数)	42955	57045	100000
ベーシックストラテジー改変 2 (理論度数)	42240	57760	
15 以上になるまでヒットする戦略 (実現度数)	42063	57937	100000
15 以上になるまでヒットする戦略 (理論度数)	42240	57760	
16 以上になるまでヒットする戦略 (実現度数)	41580	58420	100000
16 以上になるまでヒットする戦略 (理論度数)	42240	57760	
17 以上になるまでヒットする戦略 (実現度数)	40974	59026	100000
17 以上になるまでヒットする戦略 (理論度数)	42240	57760	
18 以上になるまでヒットする戦略 (実現度数)	42071	57045	100000
18 以上になるまでヒットする戦略 (理論度数)	42240	57760	
合計	295677	404323	700000

表 5.10 と表 5.11 は表 5.8 を調整したもので、負けと引き分けとを 1 つにまとめ名前を勝ち以外

とし、縦軸と横軸の合計から理論度数を計算し、付け加えたものである。

カイ 2 乗検定を行う前に必要な条件を表 5.12 でまとめる。

表 5.12　条件まとめ

帰無仮説	全戦略間の勝率に有意な差がない
対立仮説	全戦略間の勝率に有意な差がある
有意水準	5%
自由度	6
棄却値	12.59

棄却値は 12.59 である。この棄却値よりもカイ 2 乗値が大きい場合、帰無仮説を棄却して対立仮説が採択される。

カイ 2 乗値を表 5.13 に示す。

表 5.13　カイ 2 乗値

デック数無限のカイ 2 乗値	377.8012
デック数 1 のカイ 2 乗値	127.8811

デック数無限の場合、カイ 2 乗値は 377.8012 となり、カイ 2 乗値が 12.96 より大きい 377.8012 となるため、帰無仮説を棄却する。つまり、デック数無限の場合、勝率に有意な差が存在する。

デック数 1 の場合、カイ 2 乗値は 127.8811 となり、カイ 2 乗値が 12.96 より大きい 127.8811 となるため帰無仮説を棄却する。つまり、デック数 1 の場合、勝率に有意な差が存在する。

（※文責: 柿崎大輝）

デック数による勝率

カイ 2 乗検定を行うための表を作成する。まずはベーシックストラテジーで表 5.14 を作成した。

表 5.14　デック数ごとのベーシックストラテジー

	勝ち	勝ち以外 (負けと引き分け)	合計
デック数無限のベーシックストラテジー (実現度数)	42798	57202	100000
デック数無限のベーシックストラテジー (理論度数)	42955	57046	
デック数 1 のベーシックストラテジー (実現度数)	43111	56889	100000
デック数 1 のベーシックストラテジー (理論度数)	42955	57046	
合計	85909	114991	200000

ベーシックストラテジーを対象にして、デック数無限とデック数 1 の場合の勝ちと勝ち以外の 2 つを載せ、それに合計と理論度数を付け足した表である。縦軸は勝敗で横軸はデック数での戦略で

ある。シミュレータの結果から同様に他の 6 つの戦略についても表を作成した。

カイ 2 乗検定を行う前に必要な条件を表にまとめる。

表 5.15　条件まとめ

帰無仮説	デック数 1 とデック数無限の勝率との間に有意な差がない
対立仮説	デック数 1 とデック数無限の勝率との間に有意な差がある
有意水準	5%
自由度	1
棄却値	3.84

棄却値は 3.84 となる。この棄却値よりもカイ 2 乗値が大きい場合、帰無仮説を棄却して対立仮説が採択される。

ベーシックストラテジーの場合、カイ 2 乗値は 1.0665 となり、カイ 2 乗値が 3.84 より小さい 1.0665 となるため、帰無仮説を棄却しない。つまり、ベーシックストラテジーでデック数が 1 と無限では勝率に有意な差はない。これをそれぞれベーシックストラテジー改変 1、ベーシックストラテジー改変 2、15 以上になるまでヒットする戦略、16 以上になるまでヒットする戦略、17 以上になるまでヒットする戦略、18 以上になるまでヒットする戦略について同様に行う。

ベーシックストラテジー改変 1 から順にカイ 2 乗値は 4.6112、3.0753、2.2164、0.5952、0.2237、160.1305 となった。この中で棄却値 3.84 を超えたのはベーシックストラテジー改変 1 と 18 以上の場合である。ベーシックストラテジー改変 2、15 以上、16 以上と 17 以上の場合、デック数が 1 と無限では勝率に有意な差はない、ベーシックストラテジー改変 1 と 18 以上の場合、デック数が 1 と無限では勝率に有意な差はあるという結果である。

（※文責: 柿崎大輝）

5.5.2　残差分析

カイ 2 乗検定によって各戦略間の勝率に有意な差が存在することが判明した。しかし、どこに有意な差が存在するのかが分からない。そのためさらに検定を行いどの戦略に有意な差が存在するのかを探す。有意な差がどこに存在するのかを発見するため残差分析を行う。残差分析では調整済み標準化残差を算出し、調整済み標準化残差が 1.96 より大きい場合と-1.96 より小さい場合に有意な差があると分かる。調整済み標準化残差は式 5.3 で計算される。

$$\text{調整済み標準化残差} = \frac{\frac{\text{実現度数}-\text{理論度数}}{\sqrt{\text{理論度数}}}}{(1-\text{行比率})(1-\text{列比率})} \tag{5.3}$$

残差分析の手法に関しては全人類がわかる統計学 (2017) を参考に行った。

調整済み標準化残差をそれぞれ出したものを表 5.16 でまとめる。

表 5.16　調整済み標準化残差

	デック数無限		デック数 1	
	勝ち	勝ち以外	勝ち	勝ち以外
ベーシックストラテジー	7.63	-7.63	6.02	-6.02
ベーシックストラテジー改変 1	6.50	-6.50	4.73	-4.73
ベーシックストラテジー改変 2	4.00	-4.00	4.95	-4.95
15 以上になるまでヒットする戦略	5.18	-5.18	-1.22	1.22
16 以上になるまでヒットする戦略	-1.63	1.63	-4.56	4.56
17 以上になるまでヒットする戦略	-5.37	5.37	-8.75	8.75
18 以上になるまでヒットする戦略	-16.31	16.31	-1.17	1.17

表 5.16 ではそれぞれの戦略でのデック数無限と 1 の場合の勝ちと勝ち以外の調整済み標準化残差を示している。縦軸がデック数とデック数に対応した勝ちと勝ち以外の項目で、横軸がそれぞれの戦略に分かれている。表 5.16 の 1.92 以上と-1.92 以下の項目が有意な差とある判断できる。例えば、デック数無限の場合、ベーシックストラテジーの勝ちの項目を見ると値が 7.63 になっており、1.92 以上なのでベーシックストラテジーの勝ちの数はほかの戦略の勝ちより有意に多かったということが分かる。

デック数無限を見ると、ベーシックストラテジーやベーシックストラテジー改変 1、ベーシックストラテジー改変 2、15 以上になるまでヒットする戦略での勝ちの項目が 1.96 を超えておりほかの戦略より有意に多いことが分かる。逆に 17 以上になるまでヒットする戦略と 18 以上になるまでヒットする戦略は勝ちの項目が-1.96 を下回ったのでほかの戦略より有意に少ない。

デック数 1 を見ると、ベーシックストラテジーやベーシックストラテジー改変 1、ベーシックストラテジー改変 2 の勝ちの項目が 1.96 を超えておりほかの戦略より有意に多いことが分かる。逆に 16 以上になるまでヒットする戦略と 17 以上になるまでヒットする戦略は勝ちの項目が-1.96 を下回ったのでほかの戦略より有意に少ない。

ベーシックストラテジー、ベーシックストラテジー改変 1、ベーシックストラテジー改変 2 の 3 つはデック数に関係なく勝ちが有意に多いことが分かる。逆に、17 以上になるまでヒットする戦略はデック数にかかわりなく勝ちが有意に少ないことが分かる。

（※文責: 柿崎大輝）

5.5.3　多重比較

残差分析によって全戦略のどこに有意な差があるかが分かった。しかし、戦略と戦略の間に有意な差があるかはわからない。そのため、多重比較を行い、戦略間に有意な差があるかどうかを確認する。

多重比較を行う際、統計ソフトの R を使用した。R を使用する際、青木 (2010) の pairwise.prop2.test を使用した。fisher の正確確率検定を用いて、戦略間の p 値をすべて算出し、

p 値を holm 法で調整を施す。p 値が 0.05 以下の場合、有意な差がある。p 値が 0.05 より大きい場合、有意な差がないとする。それを表 5.17 と表 5.18 にまとめた。

表 5.17　デック数無限の多重比較

	ベーシックストラテジー	ベーシックストラテジー改変 1	ベーシックストラテジー改変 2	15 以上になるまでヒットする戦略	16 以上になるまでヒットする戦略	17 以上になるまでヒットする戦略
ベーシックストラテジー改変 1 勝率差	0.1					
ベーシックストラテジー改変 1 (p 値)	1.000					
ベーシックストラテジー改変 2 勝率差	0.5	0.4				
ベーシックストラテジー改変 2 (p 値)	0.109	0.522				
15 以上になるまでヒットする戦略 勝率差	0.3	0.2	-0.2			
15 以上になるまでヒットする戦略 (p 値)	0.522	1.000	1.000			
16 以上になるまでヒットする戦略 勝率差	1.3	1.2	0.8	1.0		
16 以上になるまでヒットする戦略 (p 値)	0	0	0.002	0		
17 以上になるまでヒットする戦略 勝率差	1.8	1.8	1.3	1.5	0.5	
17 以上になるまでヒットする戦略 (p 値)	0	0	0	0	0	
18 以上になるまでヒットする戦略 勝率差	3.4	3.3	2.9	3.1	2.1	1.6
18 以上になるまでヒットする戦略 (p 値)	0	0	0	0	0	0

表 5.18　デック数 1 の多重比較

	ベーシックストラテジー	ベーシックストラテジー改変 1	ベーシックストラテジー改変 2	15 以上になるまでヒットする戦略	16 以上になるまでヒットする戦略	17 以上になるまでヒットする戦略
ベーシックストラテジー改変 1 勝率差	0.2					
ベーシックストラテジー改変 1 (p 値)	1.000					
ベーシックストラテジー改変 2 勝率差	0.1	-0.1				
ベーシックストラテジー改変 2 (p 値)	1.000	1.000				
15 以上になるまでヒットする戦略 勝率差	1.0	0.8	0.9			
15 以上になるまでヒットする戦略 (p 値)	0	1.001	0.001			
16 以上になるまでヒットする戦略 勝率差	1.5	1.3	1.4	0.5		
16 以上になるまでヒットする戦略 (p 値)	0	0	0	0.158		
17 以上になるまでヒットする戦略 勝率差	2.1	2.1	2.0	1.1	0.6	
17 以上になるまでヒットする戦略 (p 値)	0	0	0	0	0.042	
18 以上になるまでヒットする戦略 勝率差	1.0	1.0	0.9	0	-0.5	-1.1
18 以上になるまでヒットする戦略 (p 値)	0	0.001	0.001	1.000	0.158	0

表 5.17 と表 5.18 はそれぞれの戦略を比較した場合の勝率差と p 値との 2 つの項目で構成されており、勝率差がプラスの場合、縦軸の戦略のほう勝率が良いことになる。例えば、縦軸がベーシックストラテジー、横軸が 15 以上になるまでヒットする戦略の部分を見ると、勝率が 1.0 で p 値が 0 となっている。p 値が 0.05 より小さいためベーシックストラテジーと 15 以上になるまでヒットする戦略の間の勝率に有意な差が存在することになり、ベーシックストラテジーのほうが勝率が 1.0 高いということになる。

デック数無限の場合、ベーシックストラテジー、ベーシックストラテジー改変 1、ベーシックストラテジー改変 2 と 15 以上になるまでヒットする戦略の 4 つで勝率に有意な差がない。16 以上に

なるまでヒットする戦略と 17 以上になるまでヒットする戦略とでは勝率に有意な差はない。それ以外には勝率に有意な差が存在した。これらを勝率の高い順に戦略を並べると、ベーシックストラテジーなどの戦略、16 と 17 以上になるまでヒットする戦略、18 以上になるまでヒットする戦略となる。これを図 5.1 で示す。

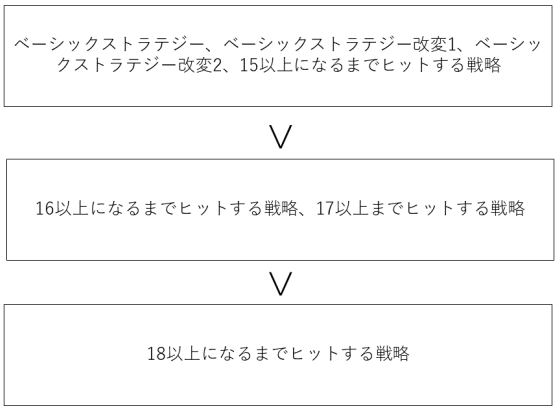


図 5.1　デック数無限の勝率順

デック数 1 の場合、ベーシックストラテジー、ベーシックストラテジー改変 1 とベーシックストラテジー改変 2 の 3 つに勝率に有意な差が存在しない。15 以上になるまでヒットする戦略、16 以上になるまでヒットする戦略と 18 以上になるまでヒットする戦略の 3 つに有意な差が存在しない。これらを勝率の高い順に戦略を並べると、ベーシックストラテジーとベーシックストラテジー改変 1 ベーシックストラテジー改変 2 の戦略、15 と 16 と 18 以上になるまでヒットする戦略、17 以上になるまでヒットする戦略となる。これを図 5.2 で示す。

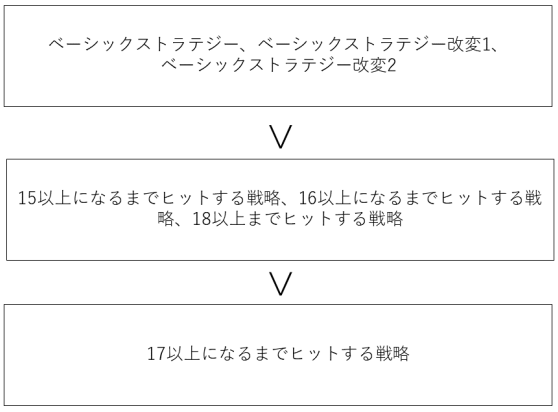


図 5.2　デック数 1 の勝率順

（※文責: 柿崎大輝）

5.6 複雑性を考慮した性能比較とその結果

5.6.1 各戦略の性能評価

今回はシミュレーションから得た各戦略の勝率と定義した複雑性を用いて、各戦略の性能比較を行った。性能の基準は以下の二通りを用意した。

性能 1. (勝率) ÷ (複雑性)

性能 2. (勝率) − (複雑性)

この評価基準に従って、デック数 1 の時の性能を表 5.19、デック数無限の時の性能を表 5.20 にまとめた。

表 5.19　デック数 1 の時の各戦略の性能

戦略	圧縮長	勝率	複雑性	性能 1	性能 2
ベーシックストラテジー	26	0.431	0.325	1.33	0.106
ベーシックストラテジー改変 1	24	0.429	0.300	1.43	0.129
ベーシックストラテジー改変 2	22	0.430	0.275	1.56	0.155
15 以上になるまでヒットする戦略	16	0.421	0.200	2.11	0.221
16 以上になるまでヒットする戦略	16	0.416	0.200	2.08	0.216
17 以上になるまでヒットする戦略	16	0.410	0.200	2.05	0.210
18 以上になるまでヒットする戦略	16	0.421	0.200	2.11	0.221

表 5.20　デック数無限の時の各戦略の性能

戦略	圧縮長	勝率	複雑性	性能 1	性能 2
ベーシックストラテジー	26	0.427	0.325	1.31	0.102
ベーシックストラテジー改変 1	24	0.424	0.300	1.41	0.124
ベーシックストラテジー改変 2	22	0.414	0.275	1.51	0.139
15 以上になるまでヒットする戦略	16	0.424	0.200	2.12	0.224
16 以上になるまでヒットする戦略	16	0.414	0.200	2.07	0.214
17 以上になるまでヒットする戦略	16	0.409	0.200	2.05	0.209
18 以上になるまでヒットする戦略	16	0.393	0.200	1.97	0.193

各戦略を比較し、次のような結果を得た。勝率のみを考慮した場合、一定の数字以上になるまでヒットする戦略よりも、ベーシックストラテジーとそれを改変した戦略の方が有意に高い勝率だった。ベーシックストラテジーと改変 1、改変 2 のそれぞれの戦略間には有意な差が見られなかった。複雑性を考慮して性能を評価した場合、デック数 1 の時は 15 以上になるまでヒットする戦略と、18 以上になるまでヒットする戦略が最も性能が良かった。デック数無限の時は、15 以上になるまでヒットする戦略が最も性能が良かった。

（※文責: 渡邊凛）

5.7　検証結果のまとめ

これまで行った検証や性能評価による結果をまとめる。
勝率のみを考慮した場合

- 結果 1:一定の数字以上になるまでヒットする戦略よりも、ベーシックストラテジーとベーシックストラテジー改変 1、ベーシックストラテジー改変 2 のほうが有意に高い勝率だった
- 結果 2:ベーシックストラテジーとベーシックストラテジー改変 1 、ベーシックストラテジー改変 2 のそれぞれの戦略間に有意な差はみられなかった
- 結果 3:ベーシックストラテジー改変 1 と 18 以上までヒットする戦略にはデック数無限とデック数 1 で勝率に有意な差があった

勝率を考慮すると、ベーシックストラテジーとベーシックストラテジー改変 1、ベーシックストラテジー改変 2 の 3 つが勝率が高く、優秀な戦略であることが分かった。ただ、ベーシックストラテジー改変 1 はデック数の違いによって勝率に差がある。
複雑性を考慮して性能を評価した場合

- 結果 4 : 15 以上になるまでヒットする戦略が 1 番優秀だった

複雑性を考慮すると、ベーシックストラテジーには改善の余地があることが判明した。複雑性の性能の基準の設定は 2 つ用意したが、基準の変化による結果の変化はなかった。

（※文責: 渡邊凛）

5.8　次の目標の設定

上記の結果を踏まえた上で、次の目標として以下のように設定した。

- デックが有限個の場合での戦略
- 賭け金の概念の導入
- より扱いやすい戦略の検証

デックが有限個の場合での戦略については、実際の対戦に従って、デック数が有限でシャッフルを一定ゲーム数まで行わず、連続でゲームを行う場合の最適な戦略を検証することを目標にしている。前述したとおりベーシックストラテジーはデック数が無限であるという前提のもと成立している戦略であるが、実際のゲームにおいてはデック数は有限であり、シャッフルを一定ゲーム数まで行わない。この事から、ゲームの進行状況により、最適とされる戦略が変わる可能性がある。そのため、デック数が有限で連続してゲームを行うと設定した状態での最適な戦略について検証する。
賭け金の概念の導入では実際のブラックジャックのゲームに則って賭け金を設定し、利得をどの

様にプラスにしていくか、そのための最適な行動を考える。前期の活動では賭け金の概念は考えず、戦略の勝率のみに着目していた。しかし、実際のブラックジャックのゲームにおいては戦略の勝率が低かったとしても賭け金の賭け方によっては利得をプラスにすることが可能である。この事から、戦略の勝率のみに着目するのではなく、賭け金の賭け方にも着目し、最終的な利得をプラスにしていく戦略を検証する。

より扱いやすい戦略の検証について、今回は複雑性の設定を手動で行い、検証する時間もあまり取らなかったため、評価基準が正確ではない可能性がある。そのため、この評価基準をどのように調整するかも検討する。

（※文責: 渡邊凛）

第 6 章　遺伝的アルゴリズム

6.1　遺伝的アルゴリズム

6.1.1　概要

本プロジェクトの目的は、ブラックジャックにおいてプレイヤーの利得を増加させ、かつ定義した複雑性がより小さくなるような戦略を発見することである。そのために我々はプレイヤーの勝率、戦略の複雑性の数値から性能 1、性能 2 という 2 つの評価値を設定した。戦略はプレイヤーの手札、ディーラーのアップカードを軸とする表で表される。評価値を大きくするような戦略を見つけることは、一般的には最適化問題と分類される。本プロジェクトで扱っているブラックジャックというゲームは、既に使用されたカードによって後のプレイにおけるカードの出現確率が変動する。そのため単純な式でゲームの進行を表現するのは現在のところ困難であり、プレイヤーの行動とゲームの結果のみを用いて戦略の最適化を行う必要がある。

また、ブラックジャックの戦略は 18 行 10 列の表で表すことができる。単純にヒット、スタンドを行うという制限を付けた場合でも、2 の 180 乗通りの戦略表が考えられる。この通り数は膨大であり、単純に全探索を行うには非現実的な時間を要する。

そのため本プロジェクトでは入出力の関係から最適なパラメータを探索するアルゴリズムのなかで、生物の進化過程を模した、遺伝的アルゴリズムという手法によって最適な戦略を発見することを試みた。遺伝的アルゴリズムは数ある探索対象の中から効率的に解を探索するため、膨大な探索範囲を持つ戦略表に対して適していると考え使用した。

（※文責: 米村祥裕）

6.1.2　アルゴリズムの説明

遺伝的アルゴリズムは生物の遺伝メカニズムをアイデアの中心として考案された最適化アルゴリズムの一つである。遺伝的アルゴリズムは次の工程からなる。

1. 初期個体群の生成
2. 各個体の環境への適応度の評価
3. 選択アルゴリズムによって次世代個体の親となる個体を抽出
4. 交叉アルゴリズムによって親個体から次世代個体を生成
5. 次世代個体に対して突然変異を適用
6. 2 から 4 を、次世代個体が必要数に達するまで繰り返し実行
7. 2 から 5 を、適応度が一定以上になるか最大世代になるまで繰り返し実行

本プロジェクトにおいて、個体とは戦略表に相当し、環境への適応度は、ブラックジャックをプ

レイした時の性能に相当する。

これから、遺伝的アルゴリズムにおいて個体がどのように表現されるのかについて遺伝子コーディングという項目で説明する。また遺伝子コーディングの説明を使って、選択アルゴリズム、交叉アルゴリズム、突然変異について説明する。

（※文責: 米村祥裕）

6.1.3 遺伝子コーディング

遺伝子コーディングとは探索対象となっている問題を遺伝子配列として扱うために対応させる作業である。生物の遺伝子には遺伝子型 (genotype) と表現型 (phenotype) とがある。遺伝子型とは遺伝子の配列のことで、表現型とは遺伝子型によって現れる形質のことである。そのため遺伝子型として存在していても発現しない形質も当然存在する。遺伝子コーディングにおいては探索対象においての一つの解パターンが表現型、解パターンを表現するための情報配列が遺伝子型に対応する。情報配列としては単純に 0 と 1 のいずれかの数値を要素としてもつバイナリ配列や順序関係を番号で表した順序配列、または文字配列などがある。巡回セールスマン問題で遺伝的アルゴリズムを適用する場合には順序配列に巡回順序をそのまま記述するなど、情報配列がそのまま表現型になる場合もある。

本プロジェクトにおいて、遺伝的アルゴリズムを使用したシミュレーションでは、ヒットとスタンドからなる戦略についてのみ探索を行ったため、バイナリ配列を用い、表現型として現れる戦略は、配列上の対応する位置の要素が 0 であればヒット、1 であればスタンドであるという変換によって遺伝子コーディングを実装した。

（※文責: 米村祥裕）

6.1.4 選択アルゴリズム

ルーレット選択方式

ルーレット選択方式とは、各個体が選ばれる確率を、適応度に比例する式で決定するアルゴリズムである。そのため適応度比例選択方式とも呼ばれる。ルーレット選択方式において、各個体 i の適応度を f_i として、個体 i が親個体として選択される確率 $P(i)$ は次の式で定義される。

$$P(i) = \frac{f_i}{\sum_k f_k}$$

エリート保存方式

エリート保存方式とは現世代における、最大の適応度である個体を次世代個体に登録する方式である。ルーレット選択では、親個体の選択が確率的に行われることから、次世代個体が現世代個体の性能を下回ることが十分に考えられる。そこで、最大の性能の個体を次世代でも作ることで性能の低下を防ぐことができる。

（※文責: 米村祥裕）

6.1.5　交叉アルゴリズム

一様交叉アルゴリズム

一様交叉アルゴリズムでは、まず個体の遺伝子長と同じ大きさのバイナリ配列を生成する。この時バイナリ配列上には 0 と 1 が並ぶが、各要素が 0 と 1 のいずれになるかはランダムに決定する。こうしてできたバイナリ配列をビットマスクという。2 つの親個体の対応する遺伝子を K_1, K_2 として, それらの遺伝子に対応するビットマスク上の要素を M と表記する。M の値に応じて次のように交叉を適用し、次世代個体 k_1, k_2 を生成する。

- $M = 0$ なら $k_1 \leftarrow K_2, k_2 \leftarrow \overline{K_1}$
- $M = 1$ なら $k_1 \leftarrow K_1, k_2 \leftarrow K_1$

遺伝子長と同じ長さのマスクを生成し、対応するマスクの値によって交叉を変化させるものであれば一様交叉と言われるため、上記のものとは異なる実装も考案されている。

（※文責: 米村祥裕）

6.1.6　突然変異アルゴリズム

個体に変化が起きなくなると、新しい解を探索することができなくなる。そのため、突然変異アルゴリズムによって個体の遺伝子情報をランダムに変化させることで防ぐことができる。突然変異アルゴリズムである。突然変異アルゴリズムでは、生成されたすべての次世代個体のすべての遺伝子配列上要素について、あらかじめ設定した確率で次の変更を行う。

- 元の要素が 0 なら 1 にする
- 元の要素が 1 なら 0 にする

（※文責: 米村祥裕）

6.1.7　実験条件

遺伝的操作についてのパラメータは表に示すように設定した。

表 6.1　遺伝的操作のパラメータ

遺伝子長	180
個体数	200
適応度	勝率 ÷ 複雑性
最大世代数	10000
交叉率	0.85
突然変異率	0.006

Beat the Dealer! Mathematics of Complex Systems and Simulation.

個体数の設定については様々な方針が提案されているが、今回は遺伝子長よりも多い数の個体を用意するというものにした。突然変異率については遺伝子長の逆数とするという方法が考案されていたため、採用した。今回の場合だと 1/180 となっている。

また、初期個体として、ベーシックストラテジーを行う個体を 5 個体、14 以上でスタンドする個体を 5 個体、15 以上でスタンドする個体を 5 個体、16 以上でスタンドする個体を 5 個体それぞれ設定した。これは、事前の実験で完全にランダムな初期個体のみだと上手く性能が向上しないということが明らかであったからである。

ブラックジャックのゲームを行うときの条件は次のように設定した。

- デック数は無限
- ゲーム数は 5 万回

ベーシックストラテジーと比較するため、前提条件として無限デックである設定にしてゲームを行わせた。

（※文責: 米村祥裕）

6.1.8 実験結果

遺伝的アルゴリズムによるシミュレーションを行った結果、図 6.1 に示すような性能の推移となった。図 6.1 中の max は最優秀性能の個体の適応度を表している。min は最低性能の個体の適応度、mean は全個体の適応度の平均値、median は全個体の適応度の中央値、top_mean は上位 10 個体の適応度の平均値、top_median は上位 10 個体の適応度の中央値を表している。

全個体の平均適応度、及び中央値はおおよそ同じように推移している。初期個体群に比べると GA の進行によって適応度は上昇している。しかし、全体の世代で通して見ると振動しているもののほぼ横ばいになっている。同様に上位 10 個体の適応度の平値、中央値も振動しているが横ばいになっている。また、上位 10 個体の適応度は初期個体群の場合と比べて大きく低下しているが、これは初期個体として与えた優秀な戦略が遺伝的操作による変更で変化したことによるものである。個体の適応度の停滞が起こっていることが全体として推察されるが、最優秀個体の適応度は順調な上昇をしているため、遺伝的操作自体はある程度成功していると考えられる。

結果として得られた最優秀な個体の戦略が表 6.2、表 6.3 である。ハードハンドでは、13 以下はすべてヒット、14 以上でスタンドという戦略になっている。ソフトハンドでは、ヒットとスタンドのみのベーシックストラテジーに対してプレイヤーの手札が A7 でディーラーのアップカードが 9 の時がヒットからスタンドに変わっており、プレイヤーの手札が A7 でディーラーのアップカードが A の時がスタンドからヒットに変わっているという形になっている。

（※文責: 米村祥裕）

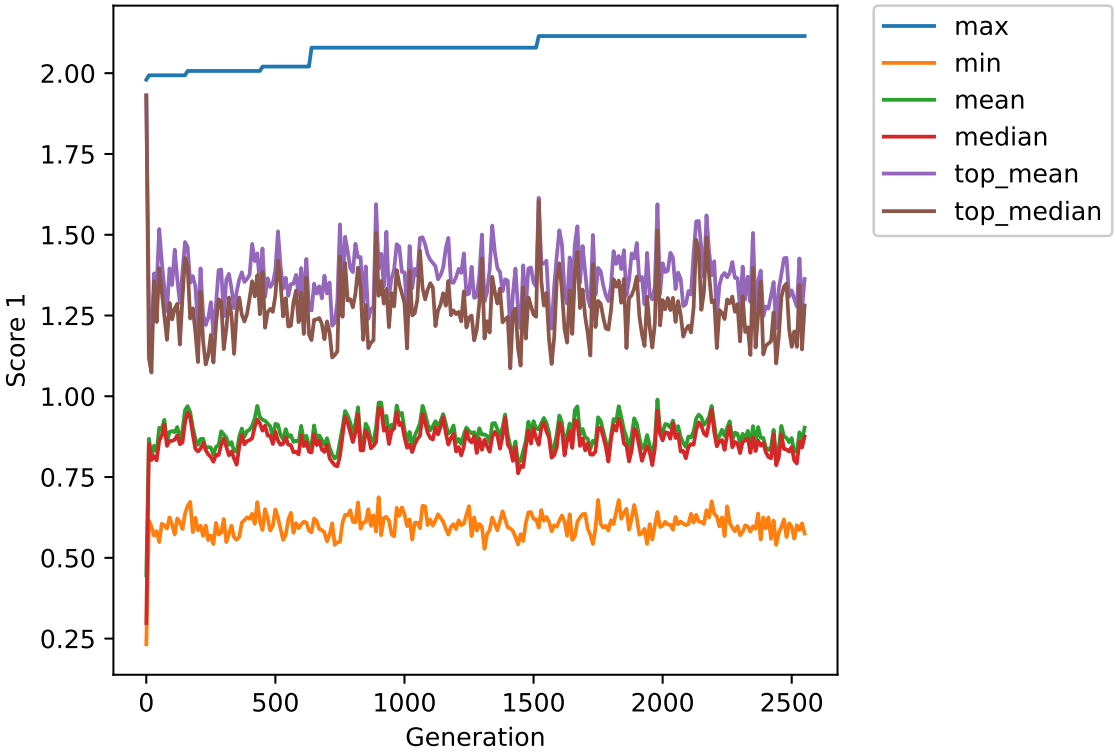


図 6.1　遺伝的アルゴリズムによるシミュレーションの進行

表 6.2　GA 戦略 (ハードハンド)

		ディーラーのアップカード									
		2	3	4	5	6	7	8	9	10	A
手札の合計	19 以上	S	S	S	S	S	S	S	S	S	S
	18	S	S	S	S	S	S	S	S	S	S
	17	S	S	S	S	S	S	S	S	S	S
	16	S	S	S	S	S	S	S	S	S	S
	15	S	S	S	S	S	S	S	S	S	S
	14	S	S	S	S	S	S	S	S	S	S
	13	H	H	H	H	H	H	H	H	H	H
	12	H	H	H	H	H	H	H	H	H	H
	11 以下	H	H	H	H	H	H	H	H	H	H

6.2　GA 戦略の戦績

6.2.1　シミュレーション結果

ここでは、遺伝的アルゴリズムから得られた戦略 (以下、GA 戦略)、ダブルダウンやスプリットが考慮されているベーシックストラテジー (以下、BS)、ヒットとスタンドのみで構成されているベーシックストラテジー (以下、BS-HS) の 3 つの戦略のシミュレーションの結果の比較について

表 6.3　GA 戦略 (ソフトハンド)

		ディーラーのアップカード									
		2	3	4	5	6	7	8	9	10	A
手札の合計	AA	H	H	H	H	H	H	H	H	H	H
	A2	H	H	H	H	H	H	H	H	H	H
	A3	H	H	H	H	H	H	H	H	H	H
	A4	H	H	H	H	H	H	H	H	H	H
	A5	H	H	H	H	H	H	H	H	H	H
	A6	H	H	H	H	H	H	H	H	H	H
	A7	S	S	S	S	S	S	S	S	H	H
	A8	S	S	S	S	S	S	S	S	S	S
	A9	S	S	S	S	S	S	S	S	S	S

述べる。なお、今回のシミュレーションではデック数は無限に設定し、10 万回ゲームを行った。その結果を表 6.4 に示す

表 6.4　デック数無限と各戦略での勝ち、負け、引き分けの比率

	勝ち（%）	負け (%)	引き分け (%)
BS	43.762	47.566	8.671
BS-HS	42.746	48.635	8.619
GA 戦略	42.858	49.502	7.64

今回は勝ち、負け、引き分けの 3 種類に分けて、比率で表したものが表 6.4 となっている。勝ちの比率が最も高いのは BS となった。そして GA 戦略は勝率では BS-HS を上回っていたが、負けの比率が最も高いのは GA 戦略となった。

（※文責: 葛西隼人）

6.2.2　カイ 2 乗検定

先程のシミュレーションを行った結果から、勝率が最も高いのは BS となったが、勝ち負け引き分けに優位な差があるのではないかと考えられる。これを確かめるために、カイ 2 乗検定を行い、勝ち負け引き分けに優位な差があるかどうかを確かめる。帰無仮説は勝ち負け引き分けに有意な差がない。対立仮説は勝ち負け引き分けに有意な差があるとし、カイ 2 乗検定を行う前に必要な条件を表 6.5 で必要な条件をまとめた。

表 6.5　勝ち負け引き分けのカイ 2 乗検定条件

帰無仮説	勝ち負け引き分けに有意な差がない
対立仮説	勝ち負け引き分けに有意な差がある
有意水準	5%
自由度	2
棄却値	5.99

棄却値は 5.99 である。この棄却値よりもカイ 2 乗値が大きい場合、帰無仮説を棄却して対立仮説が採択される。カイ 2 乗検定を行うとカイ 2 乗値は 66.754 となった。この値を p 値に変換すると 0.1 より小さくなった。よって帰無仮説を棄却して対立仮説を採択する。つまり、勝ち負け引き分けに優位な差が存在することがわかった。

（※文責: 葛西隼人）

6.2.3　残差分析

カイ 2 乗検定によって各戦略間の勝ち負け引き分けに優位な差が存在することが判明した。しかしどこに優位な差が存在するかまでは明らかになっていない。そのためさらに検定を行いどの戦略に優位な差があるかを探す。そのために残差分析を行う。残差分析では調整済み標準化残差を算出し、調整済み標準化残差が 1.96 より大きい場合と-1.96 より小さい場合に優位な差があると分かる。調整済み標準化残差をそれぞれ出したものを表 6.6 でまとめる。

表 6.6　調整済み標準化残差

	勝ち（%）	負け (%)	引き分け (%)
BS	5.051	-7.83	5.113
BS-HS	-2.98	0.59	4.277
GA 戦略	-2.016	7.293	-9.425

表 6.6 ではそれぞれ戦略での勝ち、負け、引き分けの 3 つの調整済み標準化残差を示している。例えば、BS の勝ちの項目を見ると値が 5.051 となっており、1.96 以上なので BS の勝ちの比率は他の戦略より優位に多かったということがわかる。

GA 戦略を見ると負けの比率が 1.96 を上回っており他の戦略より優位に多かったということがわかる。逆に、勝ちと引き分けの比率がどちらも-1.96 を下回っており優位に少ないことがわかった。

以上の結果から GA 戦略は勝ちの比率は BS-HS と同じだが、引き分けの比率が少なくなり、負けの比率が多くなってしまった事がわかる。

（※文責: 葛西隼人）

6.3 まとめ・手法の問題点

6.3.1 SGA の概要と問題点

この項では遺伝的アルゴリズムの問題点について説明していく。佐藤ら (1997) は代表的な遺伝的アルゴリズムでの世代交代モデルとして Simple GA(以下 SGA と表記する) を挙げ、そのモデルの問題点と改善案を示している。ただし、ここでは適応度をもとにした選択処理についてのみの問題点と改善案を挙げており、交叉アルゴリズムや遺伝子のコード設計、各種パラメータの設定に関しては扱っていない。

世代交代のモデルには、次世代の個体を生成するための親を選択する複製選択と、全ての個体の中から次世代に残す個体を選択する生存選択の 2 種類の処理が存在する。SGA の場合この 2 つの選択処理は次のようになっている。

- 複製選択
  適応度に比例した選択確率を用いたルーレット選択方式によって、集団から個体を復元抽出する。復元抽出とは一度選択された個体も次以降の選択対象に含める、つまり同じ個体が複数回選ばれることを許している選択方法である。
- 生存選択
  無条件で親集団と生成された子集団のすべてを入れ替える。

佐藤ら (1997) によると SGA には次の 3 つの問題点があることが指摘されている。

1. 高い選択圧下での早期収束
   - SGA のように適応度を用いてルーレット選択を行っている場合、探索の初期に適応度が突出した個体が存在するとその個体が複製選択において選ばれる可能性が高くなりすぎてしまう。そうなった場合、探索の序盤からその個体に遺伝子全体が収束してしまう現象が起きてしまい、最適な解にたどりつきづらくなってしまう。この現象は初期収束と呼ばれている。
2. 低い選択圧下での停滞
   - 遺伝的アルゴリズムの探索が進み、各世代の個体間の適応度に差が見られなくなってきた時に複製選択においてルーレット選択の効果が弱くなることがある。そうなると個体が最適な解に向かって進みづらくなってしまう現象が起こることがある。ここでは進化的停滞と呼んでいる。
3. 優秀な遺伝子の破壊
   - SGA では親から子に世代が移る時は無条件で全ての遺伝子を入れ替えてしまう。そのため親個体に適応度の高い個体が存在する場合でもその個体は次の世代では失われてしまう。そのため集団から適応度の高い個体が失われてしまう可能性がある。

（※文責: 伊藤晋之介）

6.3.2 SGA の改善案の紹介

前項で挙げられたように SGA の世代交代モデルには、初期収束、進化的停滞などの様々な問題点が存在している。これらの問題点を解決するため、佐藤ら (1997) では SGA の複製選択アルゴリズムと、生存選択アルゴリズムを変更したいくつかの世代交代モデルが紹介されている。

Iterated Genetic Search(IGS)

- 複製選択
  適応度を無視して集団から個体をランダムに非復元抽出する。非復元抽出とは、同じ個体を 1 度しか選択しない選び方のこと。
- 生存選択
  適応度の平均以下の個体をランダムに選び、生成された子個体のうち数体と入れ替える。

このモデルでは親個体の選択に適応度を用いず、ランダムに選択している。そのため上で挙げられた初期収束に陥る可能性を避けることができる。生存選択の時には適応度が平均以下の個体を子世代に混ぜることで、適応度が高い個体同士からは作り出せない新たな個体が生成されることを期待できる。そのため効率的に探索を行える可能性がある。

Steady State(SS)

- 複製選択
  ランキング選択法を用いて集団から個体を復元抽出する。
- 生存選択
  親集団から最悪個体を選び、生成された子個体と入れ替える。

このモデルでは複製選択にランキング選択法を用いている。ランキング選択法とは親個体を選ぶ際に適応度が高い順に個体に順位をつけていく。そして順位に応じてその個体が選択される数を決める方法。ルーレット選択方式に比べ、極端に適応度が高い個体が存在する場合でもその個体以外の個体も確実に選ばれることを保証できる。そのため初期収束してしまう可能性を抑えることができる。生存選択では IGS と同じように適応度の低い個体をいくつか混ぜることで、新たな個体が生成されることを期待できる。

CHC

- 複製選択
  適応度を無視して集団から個体をランダムに非復元抽出する。
- 生存選択
  親集団と子集団を合わせた 2 世代の中から、適応度の高い順に集団サイズ分の個体を次世代に残す。

複製選択については個体をランダムに選択し初期収束を回避している。生存選択では親個体をすべ

て子個体と入れ替えるのではなく、親世代と子世代を合わせたものの中から適応度順に集団サイズ分だけ選んでいる。この処理を行うことで、親世代の個体が次世代以降も生き残る可能性が出てくる。そうすることで親個体に存在した優秀な個体が世代交代で失われるのを防ぐことができる。

Elitist Recombination(ER)

- 複製選択
  適応度を無視して集団から個体をランダムに非復元抽出する。
- 生存選択
  各家族、すなわち親として選ばれた 2 個体とそこから生成された子の 2 個体の中から適応度の高い 2 個体を次世代に残す。

複製選択では初期収束を回避するため個体をランダムに選択している。生存選択では各家族間で適応度を比べ優秀なものが生き残るようになっている。そのため親世代の優秀な個体が失われることを防ぐことができる。また CHC とは異なり適応度の比較が家族間で行われることで、優秀な個体が急激に集団に広まることを防ぐ効果もある。

Minimal Genetic Gap(MGG)

- 複製選択
  適応度を無視して集団から個体をランダムに非復元抽出する。
- 生存選択家族の中から、最良 1 個体およびルーレット選択により選ばれた 1 個体を次世代に残す。

この設計方法は上記 4 つの改善案を基に佐藤ら (1997) が提案した手法である。SGA の問題点として挙げられている初期収束の回避のために個体選択をランダムに行っている。また生存選択の方法は上記 ER と似ているが、ER では最良 2 個体を選択していたのに対し MGG では最良の 1 個体とそれ以外からルーレット選択を行う方法を組み合わせることで適応度の分布を維持できるようにしている。

（※文責: 伊藤晋之介）

6.3.3 今回設計した GA のまとめと改善案

まとめ

今回我々が作成した遺伝的アルゴリズムでは、親世代で最も優秀な個体を次世代に残すエリート保存方式を採用し、親世代で最も優秀な 1 個体を子世代の最も成績の悪い個体と入れ替えている。そのため上記の問題点で挙げている優秀な遺伝子の破壊が起きる可能性は低いと考えられる。図 6.1 からもわかる通り最優秀個体の適応度は減少してはいない。

問題点としては、初期個体にブラックジャックの基本戦略、14 以上でスタンドする戦略、15 以上でスタンドする戦略、16 以上でスタンドする戦略をそれぞれ 5 個体ずつ設定したことが挙げられる。今回は遺伝的アルゴリズムの実行時間を短縮するため、また最適な解を早く発見するため、

戦略として優秀な基本戦略とそれを改変した戦略を初期個体に混ぜた。そのため個体全体がそれらの戦略に初期収束してしまった可能性がある。これは遺伝的アルゴリズムから出力された戦略である表 6.2、表 6.3 からもわかる。遺伝的アルゴリズムから出力された戦略表はハードハンドでは 14 以上でスタンドする戦略がそのまま出力され、ソフトハンドでは基本戦略から 1 つの遺伝子を変化させたものになっている。

改善案

改善案としては、1 つ目に初期個体に含める基本戦略などの数を減らし、初期値の補正を緩くすることが考えられる。上記の問題点でも挙げたように、今回の設計では初期値に基本戦略などを加えたことで初期収束が起きた可能性が考えられる。しかし初期値に補正を加えない場合、各個体の H と S が完全にランダムになってしまい、複雑性が高くなりすぎてしまう可能性が考えられる。そのため解の探索が上手くいかず、優秀な戦略が得られない危険性がある。

2 つ目に作成した遺伝的アルゴリズムの個体数を増やすことが考えられる。今回は遺伝的アルゴリズムの実行時間を短くしたかったことや、実行に使用したコンピュータの性能などの理由から個体数を各世代 200 体に設定した。しかしこの個体数を増やすことで一度に探索できる数が増えより良い解を見つけられる可能性がある。

3 つ目に世代数を増やし、より長く遺伝的アルゴリズムを実行するということが考えられる。前述したが今回の遺伝的アルゴリズムは実行時間を短くしている。しかし図 6.1 を見るとわかるように最優秀個体の適応度は世代を増やすごとに徐々に増加している。そのため世代数を増やし、より長く遺伝的アルゴリズムを実行することで今回得られた戦略よりも優秀である戦略が得られる可能性がある。

4 つ目にエリート保存方式の調節が挙げられる。今回エリート保存方式で保存する個体の数は各世代で 1 体のみだった。そこで保存する個体の数を増やすことで優秀な遺伝子をより多く次世代に生き残らせることができ、探索を効率的に行える効果があると考えられる。その他の改善点としては交叉確率、突然変異確率など各種パラメータの調整をすること、選択アルゴリズムをルーレット方式からランキング方式に変更すること、上記にある SGA の改善案の内容を参考にして個体選択や生存選択のアルゴリズムを変更し、様々な条件で試してみることなどが挙げられる。

（※文責: 伊藤晋之介）

第 7 章　賭け金ありのシミュレーション

ここまで勝率を評価指標としてきた。しかし、ブラックジャックにはダブルダウンやスプリットがあるので勝率が低くても利得を増やすことができる。また、賭け額を操作することでも利得を増やすことができる。例えば、10 戦して 1 勝 9 敗だとしても 1 勝の時にだけ多額を賭けていれば勝率が低くても高い利得を得ることができる。そのためここからは賭け金という概念を導入して検証を行う。

（※文責: 轟木文弥）

7.1　この章で扱うシミュレーションに用いる戦略表について

この章で扱うシミュレーションを行うにあたって、3 つの戦略を用意した。ヒットとスタンドの他に、ダブルダウンやスプリットといったルールを含めたベーシックストラテジー、ヒットとスタンドのみのベーシックストラテジー、前述の GA 戦略の 3 つである。なお、「ヒットとスタンドのみのベーシックストラテジー」は前述していたベーシックストラテジーの事を指す。

これから、3 つの戦略の内ダブルダウンやスプリットといったルールを含めたベーシックストラテジーについて説明する。ヒットとスタンドのみのベーシックストラテジー、GA 戦略については前述している為簡単に説明する。

これから説明する表の文字はそれぞれ以下のように対応している。

（H:ヒット，S:スタンド，D:ダブルダウン，P:スプリット）

ダブルダウンとスプリットについてルールの項でも触れられているが、改めて説明する。

スプリットは最初に配られた 2 枚のカードが同じ数字だった場合、最初の賭け金と同じ金額を追加して、それらのカードを 2 つに分割し、それぞれの手札で勝負するルールである。手札を 2 つに分けることで、しかし、元々手札の数字が大きかった場合にスプリットを選択すると、2 つの手札がより弱くなってしまう可能性もある。

ダブルダウンは賭け金を 2 倍にして 1 度のみヒットをし、その後強制的にスタンドするルールである。ダブルダウンを用いると次のカードで勝負が確定してしまうため、ハイリスク・ハイリターンなルールである。

どちらも、適切な場面で使用することで、より利得を増やせる可能性が上がる。

（※文責: 渡邊凛）

7.1.1　ダブルダウンやスプリットを含めたベーシックストラテジーについて

この戦略は、Thorp（1962）の 1 デック用のベーシックストラテジーではなく、Jensen（2014）の 4〜8 デック用のものを参考にしたものである。この戦略表では、スプリット、ダブルダウンが追加されている。

これらの選択肢を戦略に追加することにより、同じ賭け金でもより期待値が高くなる。ただし、選択肢が増える分表も大きくなり複雑性が増す為に人間にとって扱いづらく、誤る可能性が高いと言える。

表 7.1　ダブルダウンやスプリットを含めたベーシックストラテジー

			ディーラーのアップカード									
			2	3	4	5	6	7	8	9	10	A
手札の合計	ハードハンド	4 8	H	H	H	H	H	H	H	H	H	H
		4 8	H	H	H	H	H	H	H	H	H	H
		9	H	D	D	D	D	H	H	H	H	H
		10	D	D	D	D	D	D	D	D	H	H
		11	D	D	D	D	D	D	D	D	D	D
		12	H	H	S	S	S	H	H	H	H	H
		13	S	S	S	S	S	H	H	H	H	H
		14	S	S	S	S	S	H	H	H	H	H
		15	S	S	S	S	S	H	H	H	H	H
		16	S	S	S	S	S	H	H	H	H	H
		17 以上	S	S	S	S	S	S	S	S	S	S
	ソフトハンド	A,2	H	H	H	D	D	H	H	H	H	H
		A,3	H	H	H	D	D	H	H	H	H	H
		A,4	H	H	D	D	D	H	H	H	H	H
		A,5	H	H	D	D	D	H	H	H	H	H
		A,6	H	D	D	D	D	H	H	H	H	H
		A,7	S	D	D	D	D	S	S	H	H	H
		A,8	S	S	S	S	S	S	S	S	S	S
		A,9	S	S	S	S	S	S	S	S	S	S
		A,10	S	S	S	S	S	S	S	S	S	S
	スプリット可能	A,A	P	P	P	P	P	P	P	P	P	P
		2,2	P	P	P	P	P	P	H	H	H	H
		3,3	P	P	P	P	P	P	H	H	H	H
		4,4	H	H	H	P	P	H	H	H	H	H
		5,5	D	D	D	D	D	D	D	D	H	H
		6,6	P	P	P	P	P	H	H	H	H	H
		7,7	P	P	P	P	P	P	H	H	H	H
		8,8	P	P	P	P	P	P	P	P	P	P
		9,9	P	P	P	P	P	S	P	P	S	S
		10,10	S	S	S	S	S	S	S	S	S	S

（※文責: 渡邊凛）

7.1.2　ヒットとスタンドのみのベーシックストラテジーと GA 戦略について

ヒットとスタンドのみのベーシックストラテジーは表 1.1 をそのまま使用している。GA 戦略は遺伝的アルゴリズムによって生成された戦略表であり、表 6.2、6.3 をそのまま使用している。これらの戦略にはダブルダウンやスプリットはなく、ヒットとスタンドのみである。そのため、戦略と

しては単純になり、人間にとって扱いやすい戦略表となる。

（※文責: 渡邊凛）

7.2　一定ベット

まずは、賭け金を常に一定として、所持金がどのように変化するのかをシミュレーションで調べる。シミュレーション結果を図 7.1 と表 7.2 で示す。

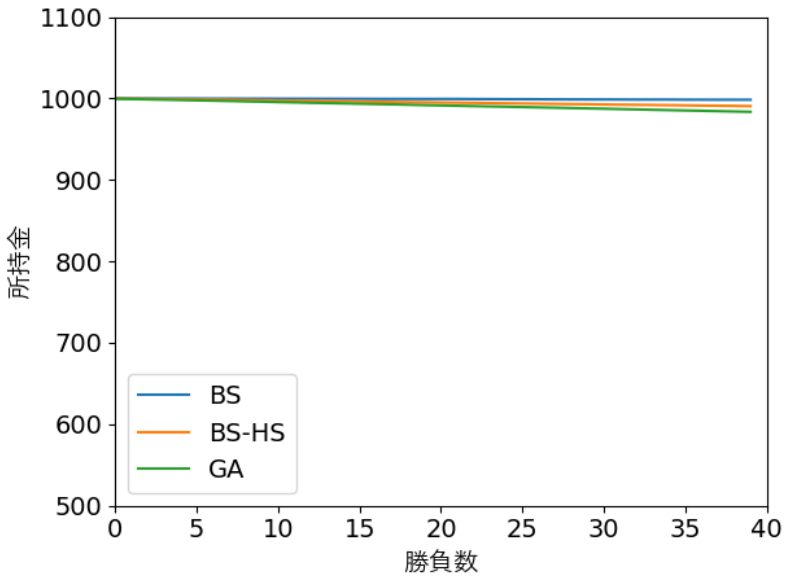


図 7.1　一定ベット

表 7.2　一定ベットの所持金

	40 回目の平均所持金	標準偏差
BS	998.328	73.538
BS-HS	990.499	62.013
GA 戦略	983.470	62.401

このシミュレーションでは最初の所持金を 1000 とし、常に 10 賭け、ブラックジャックの勝負を連続で 40 回行い、それを 5 万回行った。また戦略はダブルダウンやスプリットなどが入ったベーシックストラテジー (BS) とヒットとスタンドのみのベーシックストラテジー (BS-HS)、GA で作成した戦略 (GA 戦略) の 3 つを使用した。

図 7.1 で所持金は勝負を重ねるたびに減少していて、最後の 40 回目では最初の所持金である 1000 より少し少なくなっているように見える。そこで表 7.2 を見ると 40 回目のそれぞれの所持金が分かるが、ダブルダウンやスプリットが入ったベーシックストラテジー、ヒットとスタンドのみのベーシックストラテジー、GA 戦略の順で多いことが分かる。どの戦略でも最初の所持金である 1000 を超えることはないことが分かった。

Beat the Dealer! Mathematics of Complex Systems and Simulation.

賭け金が一定では最初の所持金を超えることができなかった。そのため本プロジェクトでは賭け金を変動させて解決を目指した。そこで賭け金を変動させる手法を 2 つ考えた。1 つ目は勝率が高くなる部分で賭け金を増加させる手法である。ブラックジャックを連続で行ったときに勝率が変化するタイミングが存在すると考え、そのタイミングで賭け金を増加させようとした。そのため、まずはブラックジャックの勝負 40 回の中で勝率が変動するところがないかを確認した。2 つ目はカウンティングという手法を用いてみる。

（※文責: 柿崎大輝）

7.2.1 勝率の推移

ブラックジャックを連続で行った場合の勝率の変化を調べる。シミュレーション結果を図 7.2 で示す。

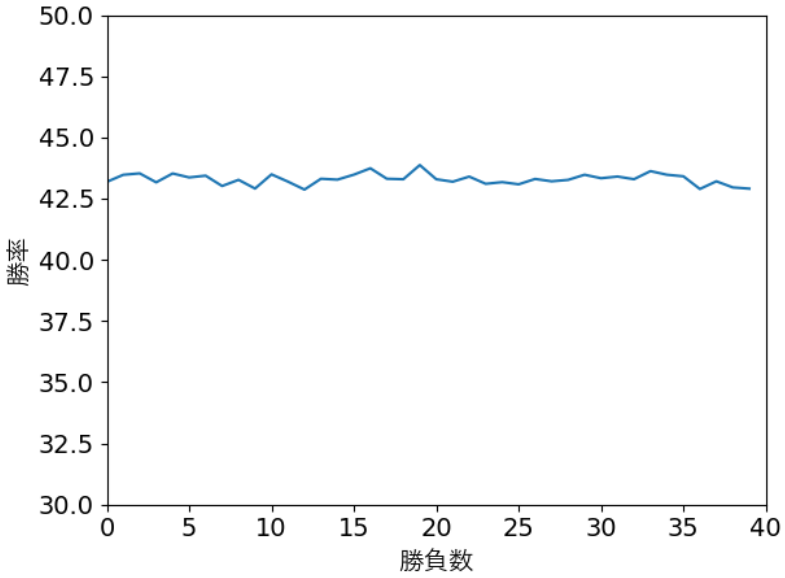


図 7.2 勝率の推移

ブラックジャックの勝負を連続で 40 回行い、それを 5 万回繰り返し、勝負ごとでの平均勝率を調べた。デック数は 6 で、使用した戦略はヒットとスタンドのみのベーシックストラテジーであった。

図 7.2 を見ると勝率は 42%〜44% の間で増減を繰り返しており、だいたい 43% の部分に集中していると分かる。勝率は特に目立った規則性は見られず、ランダムに増減を繰り返しているように見られる。勝率は最高で 43.87% で最低は 42.868% であった。

シミュレーション結果から勝率は勝負 40 回の中であまり変化しないと考えられる。また、勝率の変動に規則性は見つからなかった。そのため、勝負のどこかで賭け金を増やすという戦略はできないということが分かった。

（※文責: 柿崎大輝）

カイ 2 乗検定

40 回の勝負で 1 つ 1 つを比べてみて、勝率の差は小さいものであった。しかし、本当にその差には意味がないのか、小さくても本当は意味がある差なのではないか。それを確かめるために、カイ 2 乗検定を行い、勝率の差が有意な差であるかどうかを確かめる。

カイ 2 乗検定を行う前に必要な条件を表 7.3 でまとめる。

表 7.3　勝率の差のカイ 2 乗検定条件

帰無仮説	勝率の差に有意な差がない
対立仮説	勝率の差に有意な差がある
有意水準	5%
自由度	39
棄却値	54.57

有意水準 5% で、棄却値は 54.57 とした。カイ 2 乗値がこの棄却値 54.57 より大きければ帰無仮説を棄却し、対立仮説を採択する。

ここで、カイ 2 乗検定を行うとカイ 2 乗値は 40.207 となった。カイ 2 乗値は棄却値よりも小さくなったので、帰無仮説を採択する。よって、勝率の差には有意な差がないという結果になった。つまり、勝負 40 回で勝率の変化はないという結果となった。

（※文責: 柿崎大輝）

7.3　カウンティング

カウンティングで賭け金を変動させ、所持金を調べる。カウンティングは既存のカウンティング手法として KO 法と High-Low 法の 2 種類を用いる。そのほかの条件は一定ベットの時のシミュレーションと同じとした。シミュレーション結果を図 7.3 と図 7.4 と表 7.7 で示す。

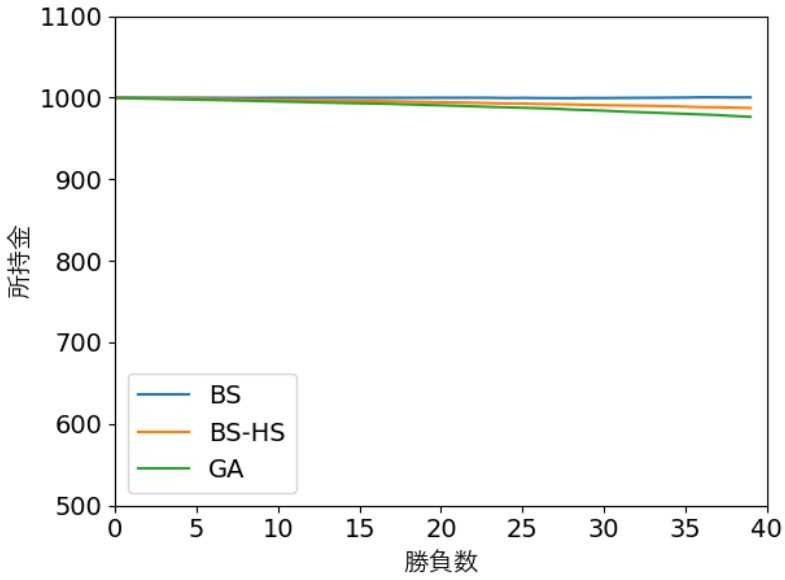


図 7.3　カウンティング KO 法

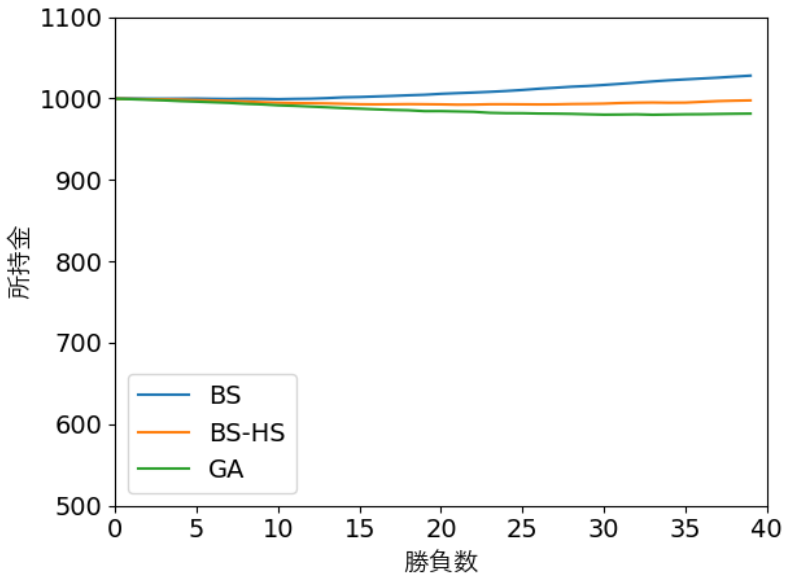


図 7.4　カウンティング High-Low 法

表 7.4　カウンティングの所持金

	40 回目の平均所持金	標準偏差
KO 法 BS	1000.237	172.770
KO 法 BS-HS	987.207	148.615
KO 法 GA 戦略	976.380	145.821
High-Low 法 BS	1027.961	298.197
High-Low 法 BS-HS	997.561	262.546
High-Low 法 GA 戦略	981.315	263.162

Beat the Dealer! Mathematics of Complex Systems and Simulation.

KO 法は図 7.3 で見ると一定ベットの時とあまり変化がないように見えるが、表 7.7 で見るとダブルダウンやスプリットが入ったベーシックストラテジーの所持金が増加していることが分かる。High-Low 法は図 7.4 ではで増えていることがはっきりと分かる。表 7.7 で見てもすべての戦略で一定ベットの時よりも増えていることが分かる。しかし、標準偏差が一定ベットや KO 法の 2 つより大きく、安定していないことが分かる。
既存のカウンティング手法を使うことでダブルダウンやスプリットが入ったベーシックストラテジーでは 1000 を超えることができた。ただ、ヒットとスタンドのみのベーシックストラテジーや GA 戦略では 1000 を超すことはできなかった。

（※文責: 柿崎大輝）

7.4　カウンティングの根拠

カウンティングが有効であるという理由は 2 つある。1 つ目に、プレイヤーは行動を選択でき、ディーラーは 16 以下の時ヒットをしなければならないというルールによるものである。例えば、デックの残りが全て 8 であり、プレイヤーはカウンティングでその状況を把握しているとする。この時、プレイヤーのハンドは 8 が 2 枚の 16 であり、ディーラーのアップカードも 8 となる。プレイヤーはここでヒットをすると 8 を引いてバーストになってしまうことが分かっているのでスタンドを選択する。一方、ディーラーの 2 枚目のカードも 8 であるため、ディーラーのハンドも 8 が 2 枚の 16 である。しかしディーラーは 16 以下のためヒットを選択する。すると 8 を引くので 24 となりディーラーはバーストになってしまう。これは極端な例だがこの場合プレイヤーの勝利が確定している。なので行動を変えられるプレイヤーにとってカウンティングは有効である。 2 つ目に、ブラックジャックにはプレイヤーに有利なカード、ディーラーに有利なカード、プレイヤーに不利なカード、ディーラーに不利なカードが存在するという点である。例えば、10 はプレイヤーに有利でディーラーに不利なカードである。ここで、10 以外のカードが多く使用されデックに 10 が多く残っている状況を想定する。この時、プレイヤーは 11 以下の時ダブルダウンを行うと 10 を引く確率が高いので 20 や 21 など 21 に近く強いハンドになり勝率が上がる。また、プレイヤーは行動を変えられるので 12 以上の時はスタンドをすることでバーストを避けることができる。さらに、ナチュラルブラックジャックの確率も上がるため利得が高くなる。反対に、ディーラーはダブルダウンの選択肢がないので利得は上がらない。また、ディーラーは 16 以下の時ヒットをするというルールがあるため、12,13,14,15,16 の時もヒットをするのでバースト率が上がる。さらにディーラーはナチュラルブラックジャックであっても配当が 1.5 倍にはならないため、利得が上がらない。以上のことから、10 以外のカードが多く使用されデックに 10 が多く残っていると、プレイヤーの利得が上がりディーラーのバースト率が上がるため 10 はプレイヤーに有利でディーラーに不利なカードであるということが言える。次に、10 が多く使用され 2,3,4 がデックに多く残っている状況を想定する。この時、プレイヤーはダブルダウンをすると 13 や 14 など 21 から遠く弱いハンドになるため勝率が下がる。また、ベーシックストラテジーではプレイヤーのハンドが 17 の時にはディーラーのハンドに関わらず全ての場合でスタンドとなっているのでプレイヤーはスタンド

を選択するが、この場合ではヒットをしてもバースト率が低いためスタンドによって勝率を下げている。さらに、ナチュラルブラックジャックの確率が低いため利得が下がる。反対に、ディーラーは元々ダブルダウンをしないため利得が下がらない。また、バースト率が下がるためディーラーの勝率が上がる。さらにディーラーはナチュラルブラックジャックであっても配当が 1.5 倍にはならないため、利得が上がらない。 以上のことから、2,3,4 はプレイヤーに不利でディーラーに有利なカードであるということが言える。また、ブラックジャックにはプレイヤーに有利なカード、ディーラーに有利なカード、プレイヤーに不利なカード、ディーラーに不利なカードが存在するということがいえる。この 2 つの理由により、カウンティングは有効であるということが言える。

（※文責: 轟木文弥）

7.4.1 カードの重要度の検証

ブラックジャックにはプレイヤーに有利なカード、ディーラーに有利なカード、プレイヤーに不利なカード、ディーラーに不利なカードが存在する。そこで本プロジェクトは、どのカードがどれほどプレイヤーにとって有利なカードかを検証した。検証方法として、普通のデ 1 ック、1 デックから 1 種類カードを抜いたデックをそれぞれ用意し、シミュレーションした結果を比較した。比較対象となるデックは以下の通りである。

- 普通の 1 デック
- A を抜いたデック
- 2 を抜いたデック
- 3 を抜いたデック
- 4 を抜いたデック
- 5 を抜いたデック
- 6 を抜いたデック
- 7 を抜いたデック
- 8 を抜いたデック
- 9 を抜いたデック
- 10 を抜いたデック

7.4.2 仮説

10,A が残っていると勝率が上がる。逆に 2,3,4 が残っていると勝率が下がる。5,6,7,8,9 は影響がないという仮説を設定する。これは次の 3 つに注目したからである。1 つ目に、数値の小さいカードはダブルダウンの勝率が下がり、ディーラーのバースト率が下がることからプレイヤーにとって不利である。2 つ目に、10 はダブルダウンの勝率が上がり、ディーラーのバースト率が上がることからプレイヤーに有利である。3 つ目に、A はプレイヤーの方が 1 と 11 を自由に選択でき、ナチュラルブラックジャックの可能性もあるため有利である。この 3 つの考えから、仮説を設定

Beat the Dealer! Mathematics of Complex Systems and Simulation.

した。

（※文責: 轟木文弥）

7.5　シミュレーション

設定した仮説をそれぞれの数字をデックから抜いた場合のシミュレータを回して勝率を確認した。これによりカードの重要度を測った。デック数は 6 とした。戦略はダブルダウン、スプリット、サレンダーを含むベーシックストラテジーを使用した。シミュレーション回数は 10 万回とした。ただし、今回は賭け金を導入しなかったためダブルダウンは利得が 2 倍になるという点を勝利数 2 とすることで表現した。シミュレーション結果を 7.5 に示す。

表 7.5　カードを抜いたデックの勝率

	勝ち (%)	勝ち以外 (引き分けと負け、%)
普通の 1 デック	42.653	57.347
A を抜いたデック	41.576	58.424
2 を抜いたデック	42.697	57.303
3 を抜いたデック	43.023	56.977
4 を抜いたデック	42.891	57.109
5 を抜いたデック	43.940	56.060
6 を抜いたデック	43.756	56.244
7 を抜いたデック	42.851	57.149
8 を抜いたデック	42.047	57.953
9 を抜いたデック	42.127	57.873
10 を抜いたデック	41.576	58.424

8,9,10,A を抜いたデックは普通のデックより勝率が下がり、2,3,4,5,6,7 を抜いたデックは勝率が上がった。得られたデータをカイ 2 乗検定と多重比較を用いてどこに差が存在するかを検証した。結果として、5,6,10,A には有意な差が存在することが確認された。つまり、5,6 はプレイヤーに不利なカードであり、10,A はプレイヤーに有利なカードであるという結果が得られた。

（※文責: 轟木文弥）

7.6　新しいカウンティング手法の提案

KO 法は安定しているが変化が小さく、High-Low 法は変化が大きいが安定しなかった。本プロジェクトは安定していて変化が大きいカウンティング手法を見つけるべく新たに 2 つのカウンティング手法を考えた。それが RUKO 法と 5-6-A 法である。今回は KO 法をベースにして、安定している KO 法の変化を大きくする手法を考えた。ここではその 2 つのカウンティング手法について詳しく説明する。

（※文責: 轟木文弥）

7.6.1 RUKO 法

まず RUKO 法について説明する。RUKO 法とは Risk Up KO 法の略で、KO 法のリスクを上げることで利得を上げようとしたカウンティング手法である。KO 法は 6 デックの場合初期ランニングカウントが-20 で、ランニングカウントが-5 以下の時は 1 ユニットしか賭けない。そのため賭け金を上げる段階が遅くなっている。そこで RUKO 法は、初期ランニングカウントを KO 法の半分とした。すなわち 6 デックの場合は-10 である。これにより KO 法より早い段階からランニングカウントが-5 を超えることで賭け金を大きくし、利得を上げようとした。カードカウントとベットシステムは KO 法と同じである。

（※文責: 轟木文弥）

7.6.2 5-6-A 法

次に 5-6-A 法について説明する。5-6-A 法とは、カードの有利さの検証結果から有意な差があった部分の 5,6,A を考慮したカウンティング手法である。先述のように、KO 法は賭け金を上げる段階が遅くなっている。そこで 5-6-A 法では以下の 7.6.2 のように、特に不利なカードである 5 と 6 が出た時ランニングカウントを +2 し、特に有利なカードである A が出た時ランニングカウントを-2 した。ただし、10 は特に有利なカードであるが、10 はデックに存在する枚数が多くランニングカウントを-2 するとカウントが大きくならず賭け金があまり上がらなくなる。これは賭け金を上げるという意図に反しているため 10 が出てもランニングカウントを-1 とした。

表 7.6　5-6-A 法でのカードカウント

2	3	4	5	6	7	8	9	10	A
+1	+1	+1	+2	+2	+1	0	0	-1	-2

初期ランニングカウントとベットシステムは KO 法と同じである。この手法により KO 法よりランニングカウントを大きくし賭け金を上げ利得を増やそうとした。また、このカードカウントにより正確にデックの状態を判断しようとした。

（※文責: 轟木文弥）

7.7 RUKO 法、5-6-A 法のシミュレーション

カウンティングで賭け金を変動させ、所持金を調べる。カウンティングは本プロジェクトで作成したカウンティング手法の RUKO 法と 5-6-A 法の 2 種類を用いる。そのほかの条件は一定ベットの時のシミュレーションと同じとした。シミュレーション結果を図 7.5 と図 7.6 と表 7.7 で示す。

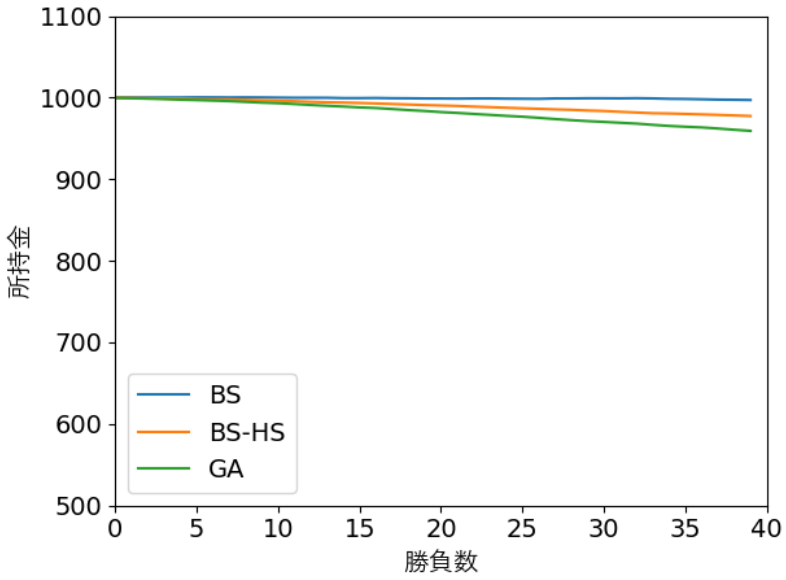


図 7.5　カウンティング RUKO 法

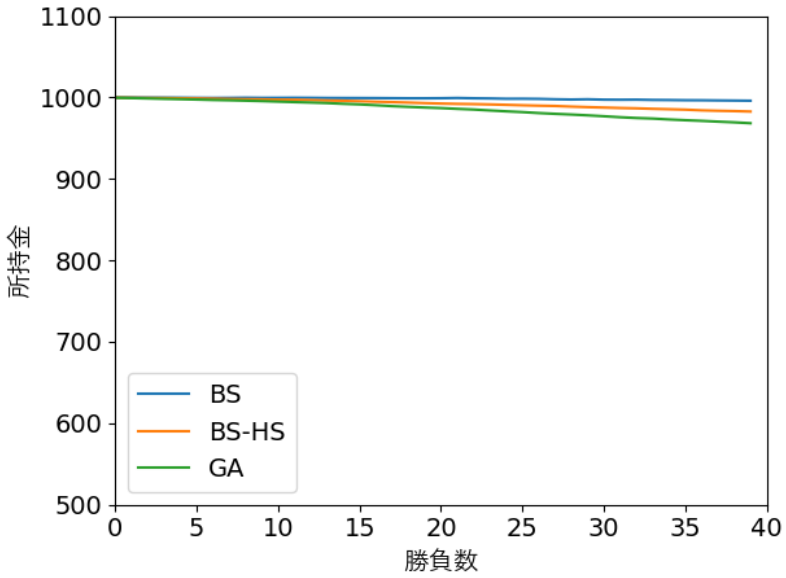


図 7.6　カウンティング 5-6-A 法

表 7.7　カウンティングの所持金

	40 回目の平均所持金	標準偏差
KO 法 BS	1000.237	172.770
KO 法 BS-HS	987.207	148.615
KO 法 GA 戦略	976.380	145.821
High-Low 法 BS	1027.961	298.197
High-Low 法 BS-HS	997.561	262.546
High-Low 法 GA 戦略	981.315	263.162
RUKO 法 BS	1000.237	172.770
RUKO 法 BS-HS	987.207	148.615
RUKO 法 GA 戦略	976.380	145.821
5-6-A 法 BS	995.918	211.884
5-6-A 法 BS-HS	982.640	179.584
5-6-A 法 GA 戦略	968.303	180.464

（※文責: 轟木文弥）

7.8　RUKO 法のまとめ

RUKO 法の特徴は、6 デックの場合の KO 法の初期ランニングカウントが-20 であるのに対し-10 に変更することで、より早い段階から賭け金を大きくすることにより最終的な利得を増やすことを目的として作成したものである。RUKO 法を使用した場合の 40 回目の所持金の推移のグラフ 7.5 を見てみると、ベーシックストラテジー、ヒットとスタンドのみのベーシックストラテジー、GA 戦略の全ての戦略が最初の所持金額 1000 を下回る結果となった。これはオリジナルの KO 法と比較してみても、より平均所持金を下回っていたことがわかった。また、標準偏差も KO 法と比べて増加しており、安定していないことがわかった。

（※文責: 薩田凱斗）

7.9　5-6-A 法のまとめ

5-6-A 法の特徴は、KO 法のカードカウントを変更することで、より早い段階から賭け金を大きくすることにより最終的な利得を増やすことを目的として作成したものである。5-6-A 法を使用した場合の 40 回目の所持金の推移のグラフ 7.6 を見てみると、ベーシックストラテジー、ヒットとスタンドのみのベーシックストラテジー、GA 戦略の全ての戦略が最初の所持金額 1000 を下回る結果となった。これはオリジナルの KO 法と比較してみても、より平均所持金を下回っていたことがわかった。また、標準偏差も KO 法と比べて増加しており、安定していないことがわかった。

（※文責: 轟木文弥）

7.10　RUKO 法と 5-6-A 法のまとめ

既存のカウンティング手法である KO 法と High-Low 法の一部を変更して作成した RUKO と 5-6-A の結果をまとめると、どちらもぞれぞれのオリジナルの手法よりも安定性が下がり平均所持金が減るという悪い結果となってしまった。今後の課題としては、それぞれの手法で改変した部分がなぜ平均所持金を下回った結果となってしまったかを究明するとともに、目的である最終的な利得を増やすことに重点を置いて改良していきたい。

（※文責: 薩田凱斗）

7.11　カウンティング手法全部のまとめ

ここで注目するのは GA 戦略である。一定ベットやカウンティングを使用しても所持金がすべての戦略で一番少なかった。GA 戦略は遺伝的アルゴリズムで探索し、発見した戦略であるのになぜ他の戦略より悪い結果となったのか。それは複雑性ということを考慮していないからではないかと考えた。GA 戦略は勝率と複雑性からなる性能で探索したもので、行ったシミュレーションでは複雑性を考慮する部分がなく、その結果一番悪い結果となったと考えた。そこで複雑性を確かめる実験の結果を用いて、”エラー率”という指標を作成し、シミュレーションに導入して行うこととした。

（※文責: 柿崎大輝）

7.12　エラー率の導入

GA 戦略が結果として最も低い所持金となった原因は、シミュレーションを行う条件にあった。GA 戦略は複雑性が低い、つまりシンプルであることが最大の特徴である。そこで、我々は GA 戦略の特徴がどのくらい有用かを調べるために複雑性を考慮したシミュレーションを行う必要があった。そして、そのシミュレーションを行うため、複雑性に関する実験の結果をもとにエラー率という概念を作成した。エラー率の定義は、ある戦略を使用する際にプレイヤーが行動を間違えてしまう確率を表したものである。このエラー率をシミュレーションに適用することにより、GA 戦略を使用した際のプレイヤーの所持金が増加する可能性があるのではないかと考えた。このエラー率の導出方法として、複雑性に関する実験データの正答率をもとに線形回帰分析を行った。また、使用した実験データについては以下の表 7.8 で示す通りである。

表 7.8　複雑性と誤答率

複雑性	0.2	0.2	0.407	0.448
エラー率	0.002	0.009	0.125	0.162

また、以上の実験データから線形回帰分析により得られた式は、

$$y = 0.611x - 0.1172 \tag{7.1}$$

となった。この式を横軸を複雑性、縦軸をエラー率としてグラフにしたものが図 7.12 である。

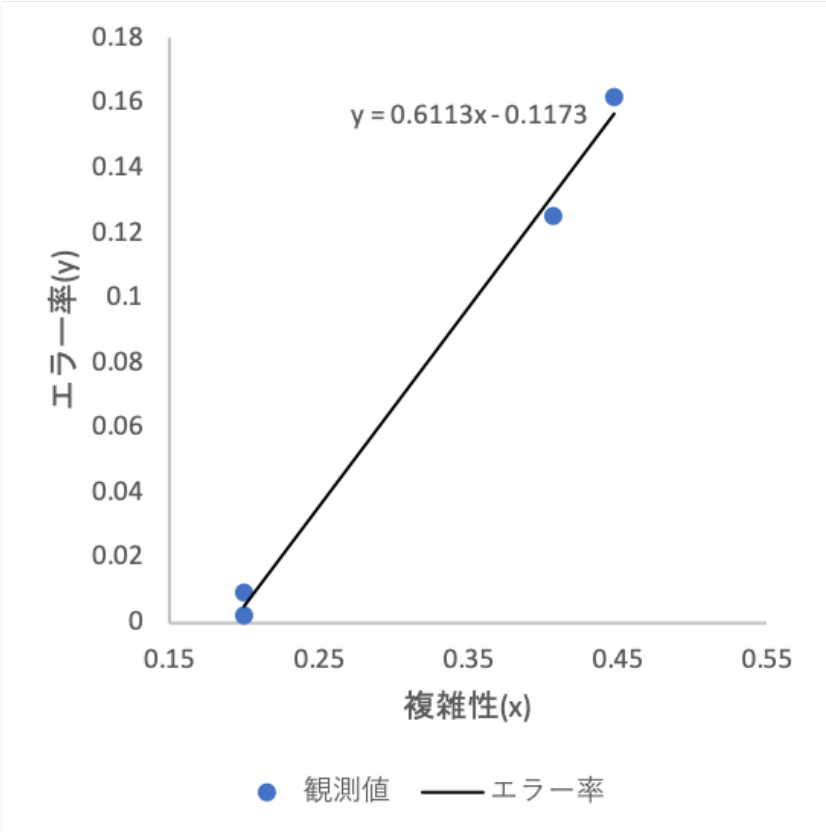


図 7.7　線形回帰分析によるエラー率のグラフ

（※文責: 薩田凱斗）

7.13　エラー率を導入したシミュレーション

エラー率を導入して、シミュレーションを行う。エラー率は戦略に従って行動するときにミスをする確率で、ミスをすると戦略表とは異なる行動を実行するようにした。またカウンティング手法は KO 法と High-Low 法の 2 種類を用いた。それ以外の部分は前のシミュレーションと同じ条件とした。表 7.9 が戦略ごとのエラー率で、図 7.8 と図 7.9、表 7.10 がシミュレーション結果である。

表 7.9　戦略ごとのエラー率

戦略	エラー率 (%)
BS	16.2
BS-HS	4.8
GA 戦略	0.5

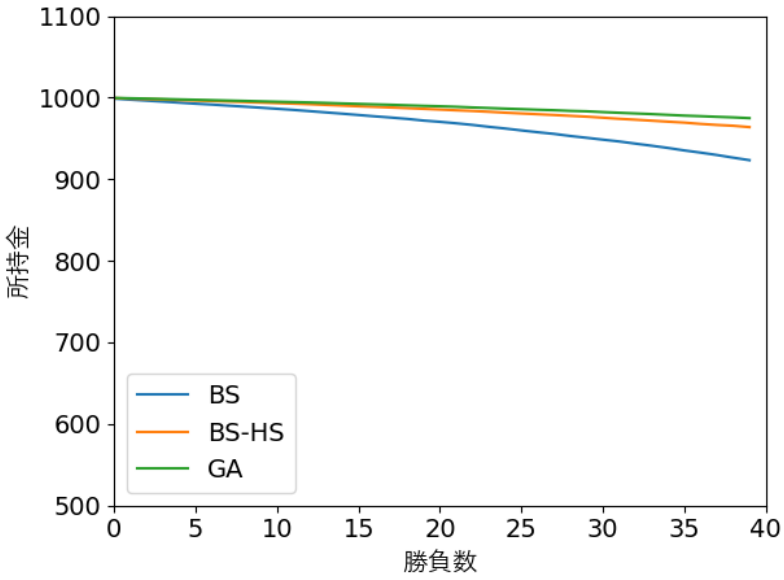


図 7.8　エラー率ありの KO 法

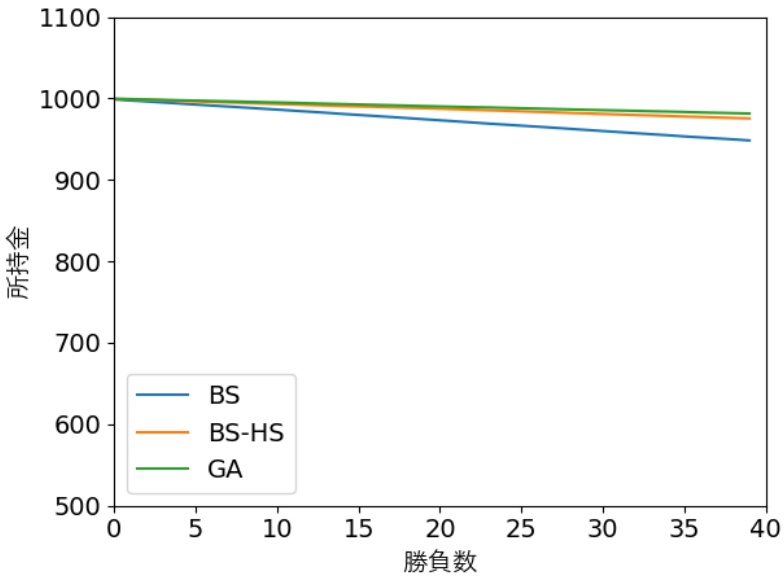


図 7.9　エラー率ありの High-Low 法

表 7.10　エラー率を使用した際の所持金

	40 回目の平均所持金	標準偏差
KO 法 BS	923.303	174.255
KO 法 BS-HS	963.799	150.135
KO 法 GA 戦略	974.817	144.821
High-Low 法 BS	948.394	79.764
High-Low 法 BS-HS	975.440	67.263
High-Low 法 GA 戦略	981.474	67.747

Beat the Dealer! Mathematics of Complex Systems and Simulation.

エラー率を導入した結果では GA 戦略、ヒットとスタンドのみのベーシックストラテジー、ダブルダウンやスプリットが入ったベーシックストラテジーの順となり、所持金が 1 番多かったダブルダウンやスプリットが入ったベーシックストラテジーと 1 番少なかった GA 戦略が入れ替わった。特にダブルダウンやスプリットが入ったベーシックストラテジーはエラー率を導入することで大きく所持金が少なくなった。またエラー率を導入したシミュレーションでは 1000 を超える戦略はないことが分かった

エラー率を導入した場合、GA 戦略が 1 番優れていることが分かった。しかし、GA 戦略では 1000 を超えることができないことも分かった。この点に関して GA 戦略は改善がする必要があると考えられる。

（※文責: 柿崎大輝）

7.13.1　分散分析

エラー率を導入したシミュレーションの時、3 つの戦略間で 40 回目の平均所持金に有意な差があるかどうかを調べる。まず最初に有意な差が存在するかどうかを調べる。存在すればまた別の手法でどの戦略間に有意な差が存するのかを調べる。

3 つの戦略において有意な差があるかどうかを調べるため、分散分析を用いた。分散分析とは 3 群以上のデータにおいて平均の差を検定する手法である。この分散分析を KO 法と High-Low 法の 2 種類のデータで行った。分散分析の詳しい条件は以下の表 7.11 にまとめた。なお、分散分析を行うにあたって統計ソフトの R を使用した。

表 7.11　分散分析の条件

帰無仮説	3 つの戦略で平均所持金に有意な差はない
対立仮説	3 つの戦略で平均所持金に有意な差はある
有意水準	5%

分散分析を行うと、KO 法、High-Low 法の両方で p 値がとても小さくなり、0.05 以下となる。よって帰無仮説を棄却し、対立仮説を採択する。つまり、3 つの戦略で有な差が存在することが確認できた。次は 3 つの戦略間のどこに有意な差が存在するのかを調べる。

（※文責: 柿崎大輝）

7.13.2　多重比較

3 つの戦略間のどこに有意な差が存在するのかを調べるために多重比較を用いた。有意水準を 5% として行った。結果は表 7.12 と表 7.13 に示す。

表 7.12　KO 法での多重比較

	BS	BS-HS
BS-HS 所持金の差	40.496	
p 値	0	
GA 所持金の差	51.515	11.019
p 値	0	0

表 7.13　KO 法での多重比較

	BS	BS-HS
BS-HS 所持金の差	27.046	
p 値	0	
GA 所持金の差	33.080	6.034
p 値	0	0

表 7.12 と表 7.13 ではそれぞれの戦略を比較した場合の所持金の差と p 値を表示している。所持金がプラスの場合は列側の戦略のほうが所持金が多く、逆にマイナスなら行側の戦略のほうが所持金が多い。また、p 値が 0.05 より小さい場合、その 2 つの戦略で有意な差が存在するということが分かる。

KO 法ではすべての戦略間で p 値が 0.05 以下となり、すべての戦略間で有意な差が存在することが分かった。High-Low 法でもすべての戦略間で p 値が 0.05 以下となり、すべての戦略間で有意な差が存在することが分かった。これで KO 法 High-Low 法の両方ですべて戦略間で有意な差が存在する。

（※文責: 柿崎大輝）

第 8 章　まとめ

まず、本プロジェクトは、複雑性の実験により次の結果を得た。

- 結果 1:複雑性とテストの成績には強い負の相関がある
- 結果 2:CRT の正答率とテストの成績には相関がない
- 結果 3:リスク回避性とテストの成績には相関がない

結果 1 から、複雑性が高くなると戦略を間違えやすいため本プロジェクトが定義した複雑性は評価指標として適しているということが判明した。また、結果 2 と 3 からブラックジャックの戦略を扱う能力と一般的な認知能力は関係がないということが判明した。

次に、ベーシックストラテジーについて比較対象となる 6 つの戦略を考案し、作成したブラックジャックシミュレータで検証を行った。これにより次の結果が得られた。

- 結果 1:一定の数字以上になるまでヒットする戦略よりも、ベーシックストラテジーとベーシックストラテジー改変 1、ベーシックストラテジー改変 2 のほうが有意に高い勝率だった
- 結果 2:ベーシックストラテジーとベーシックストラテジー改変 1 、ベーシックストラテジー改変 2 のそれぞれの戦略間に有意な差はみられなかった
- 結果 3:ベーシックストラテジー改変 1 と 18 以上までヒットする戦略にはデック数無限とデック数 1 で勝率に有意な差があった

これらの結果から、勝率を考慮すると、ベーシックストラテジーとベーシックストラテジー改変 1、ベーシックストラテジー改変 2 の 3 つが勝率が高く、優秀な戦略であることが分かった。ただし、ベーシックストラテジー改変 1 はデック数の違いによって勝率に差があった。さらに、複雑性を考慮して性能を評価した場合次の結果が得られた。

- 結果 4 : 15 以上になるまでヒットする戦略が 1 番優秀であった

よって、複雑性を考慮すると、ベーシックストラテジーには改善の余地があることが判明した。

次に、遺伝的アルゴリズムを用いて新しい戦略を得た。得られた戦略とベーシックストラテジー、ダブルダウンやスプリットを含めたベーシックストラテジーと勝率を比較した。複雑性を考慮して新しい戦略を作成したが、比較の際には複雑性を考慮しなかったため、最も成績が悪いという結果となった。これにより複雑性を考慮したシミュレーションが必要ということが判明した。

次に、賭け金を導入してベーシックストラテジー、ダブルダウンやスプリットを含めたベーシックストラテジー、遺伝的アルゴリズムでえられた戦略で検証を行った。まずは賭け金を一定にした場合の所持金を調査した。結果として全ての戦略が初期所持金を下回った。次にカウンティングを用いることで利得を増やすことを試みた。結果として、既存のカウンティング手法を使うことで初期所持金を上回ることに成功した。次に新しいカウンティング手法を考案し検証した。結果として

既存の手法より良い結果を得られることができなかった。よってここからは既存のカウンティング手法を使用することとした。

次に、複雑性を考慮してシミュレーションをするために、複雑性の実験データをもとにエラー率という概念を作成した。エラー率を適用してシミュレーションを行った結果、本プロジェクトが作成した、遺伝的アルゴリズムを用いて得られた戦略が最も優秀であるという結果が得られた。

（※文責: 轟木文弥）

第 9 章　今後の課題

我々が一年を通して活動した結果、今後の課題は以下の 2 点であると判断した。

- 利得をプラスにできる戦略の探索
- ブラックジャックをプレイする際に重要な能力の調査

このプロジェクトで最終的に一番優秀と判断された戦略を用いてシミュレーションをしたところ、最終的なプレイヤーの利得はマイナスであった。また、賭け金の賭け方からアプローチするため、High-Low 法と KO 法というカウンティング手法を、基本戦略、ヒットとスタンドのみの基本戦略、今回導出された GA 戦略の 3 つの戦略を用いて賭け金のシミュレーションを行ったが、いずれも利得がプラスになることはなかった。

我々のプロジェクトの目的は、ディーラーをやっつける、すなわちカジノ側から利益を得ることが大本の目的であるため、まだ目的を達成したとは言えない。

なぜこのような結果となったのか、我々の予想としては、今回の遺伝的アルゴリズムのシミュレーションにはエラー率という概念が導入されていないためと思われる。

複雑性は単にその戦略の複雑さを数値にしただけであるが、エラー率はそれに加え、人間がどれだけ戦略を間違えやすいかという要素が含まれている。

戦略の勝率を導出するために作成したシミュレータには、エラー率の概念が導入されているが、その戦略を探索する遺伝的アルゴリズムのプログラムには導入されていなかった。つまり、戦略を作成する段階と、その戦略を評価する段階で、評価指標が異なるものであったのである。そのため、作成段階では優秀とされた戦略が、評価するにあたって性能が良くないと判断されてしまったのである。

以上のことより、利得をプラスにするためには、勝率が 50 %以上の戦略を見つけだしゲームを行うか、カウンティングの手法を改変して利益を出すようにしなければならない。

そこで今後の課題として、遺伝的アルゴリズムにエラー率の概念を導入し、より勝率の高い戦略を見つけ出すことと、カウンティングの手法の中で優秀なものを、今回使用した遺伝的アルゴリズム、もしくは他の方法を用いて探索する事が挙げられた。

また、今回の複雑性の検証実験により、ブラックジャックをプレイする際、戦略表を記憶する上で、一般的な認知的判断能力は関係がないことがわかった。つまり、別の指標を用いて、戦略表を記憶する能力を測ることができる可能性があるということである。その能力とは何であるのかを調査することが 2 つ目の課題として挙げられた。

（※文責: 菱田美紗紀）

第 10 章　前期活動

本プロジェクトで題材としているブラックジャックというゲームを実際のプレイも交えて学習した。前章までで説明したベーシックストラテジーを試すということもその中で行ったが、実際に使用してみると表を覚え、かつ素早いゲームの進行に合わせながら実行するのは容易ではないという実感が得られた。もちろんディーラー側に戦略の実行をさとられないようにするためには、ゲームの進行を止めるなど違和感を持たせるような行動はできるだけ無くす必要がある。以上のことから、戦略の単純化の必要性を再確認した。

シミュレータは Python3 で作成を行った。前期までで得られたブラックジャックの戦略比較に用いた数値はこのシミュレータによって得られた。また開発の効率化のためにバージョン管理システムである Git を導入し、Git によるバージョン管理について学習した。導入の理由としては、プロジェクト学習でプログラム開発を行う場合複数人で作業を行うためである。複数人の作業では、最新版のソースコードを作業者同士で共有する必要がある。単純にプログラムファイル群をひとつにまとめて送付し合うという方式もできるが、これではどの部分が変更されたのかの確認を行いにくい。Git というシステムでは、誰がどのようなタイミングでどのような変更を行ったかが記録されるため、エラーの解消を効率的に行うことができる。

また、中間報告書の作成作業にも Git を用いた。中間報告書の記述では、プログラムのソースコード以上に文責者を明確にすることがプロジェクト学習として要求されている。また、報告書の記述は TeX であり、テキストファイルであるため、Git で管理することができる。単純にファイルをまとめて送付する場合よりも円滑な管理が行えたと思う。

以上のように、導入したバージョン管理システム Git はプロジェクト学習の多くの作業で有効だった。

シミュレータの正しさ、シミュレーション結果の分析のために統計学を学習した。

本プロジェクトにおいて、戦略の複雑性を評価することはとても重要な事項である。複雑性の定義付けのために Chaitin(1969) によって定義されたコルモゴロフ複雑性の定義を参考にし、コルモゴロフ複雑性の定義と使われ方について調査し学習した。

本プロジェクトでは最適な戦略の探索を行うための技術の一つとしてニューラルネットワークを挙げ、斎藤 (2016) の書籍を教科書とし、勉強会を行った。

（※文責: 米村祥裕）

第 11 章　後期活動

後期は、まず主に 3 つの課題を設定し、それぞれの課題解決に向かってグループを作成し活動した。

１つ目の課題は、遺伝的アルゴリズムを使用して新しいブラックジャックの戦略を探索することである。以下、この課題解決を行ったグループ名を、GA 班とする。

２つ目の課題は、前期活動で設定した複雑性という要素が、人間が戦略表を記憶する能力と一致しているのかを、実験を通して検証することである。以下、この課題解決を行ったグループ名を、複雑性班とする。

３つ目の課題は、最適な戦略の探索に、賭け金の概念を導入し、それに伴いカウンティングについても既存のものを学習し、賭け方に考慮することである。以下、この課題解決を行ったグループ名を、カウンティング班とする。

以下に、それぞれのグループの活動のあらましを述べる。

- GA 班

まずはじめに、遺伝的アルゴリズムを用いて戦略を探索するプログラムを作成した。遺伝的アルゴリズムを使用した背景には、ブラックジャックの戦略の組み合わせは膨大であり、遺伝的アルゴリズムを使用するのが最適であると判断したという理由がある。使用した言語は Python である。探索は、研究費で購入した計算用 PC を使用し、また学内から接続できるようにネットワーク設定も行った。

そのようにして得られた戦略を用いて勝率を導出したが、予想されていたほどの結果は出なかった。そこで原因を解明したところ、交叉手法や突然変異確率、初期個体値の見直しが必要であると判明した。それらを見直したうえで最終的に GA 戦略という新たな戦略を導出した。

- 複雑性班

まずはじめに、コルモゴロフ複雑性を参考にし、複雑性とカウンティングなどの戦略の組み合わせで利得が最大になるような戦略の評価方法を考えた。

その結果、戦略の評価方法を知るには実際に人間に戦略表を記憶させる実験を行うことが必要だと判明したため、実験の計画をたてた。実験には認知的判断能力も知る必要があったため、知能テストの作成も行った。

実験から得られた結果をもとに、前期に作成した複雑性との相関を検定した。その結果、２つの間には非常に強い負の相関がみられた。つまり、複雑性が高くなるにつれ、戦略表を間違えて覚えやすいということであり、我々が導出した複雑性は評価指標として適していることが明らかになった。

また、同時に行った認知的判断能力テストの結果から、ブラックジャックの戦略を記憶するうえ

Beat the Dealer! Mathematics of Complex Systems and Simulation.

で一般的な認知能力は関係がないという新しい結果も得られた。

- カウンティング班

まずはじめに、前期に作成したシミュレータを改変し、ダブルダウン、スプリット、サレンダーが行えるようにした。それに伴い、それらのルールを織り込んだカウンティングとベッティングシステムについて調査した。ベッティングシステムについては、それぞれの比較が必要であったため、比較方法の手法の検討も行った。

これらの研究から、ある戦略を使用した際、膨大な数のゲーム数を行った場合の最終的な利得を、カウンティング手法を変更した時でも導出できるシミュレータの開発に成功した。そしてこのシミュレータを使用して、GA 班が導出した戦略について検証した。

ところが GA 班から導出された戦略を用いた戦略では、基本戦略を超えた結果が出ることはなかった。これについて検討したところ、戦略を間違えて覚える確率 (以下、エラー率とする) について考えていなかったためそのような結果になったと考えられた。そこで、複雑性班が行った実験の結果を基にエラー率を導出し、シミュレーションに組み込んだ。その結果、当初の予想の通り、GA 班が導出した戦略が一番優秀であるという結果が得られた。

- 共通して行った事

函館市の高校生が参加する見学会があったため、そのためのスライド作成などを行った。12 月には、一年の活動の総まとめとして成果発表会に参加した。

（※文責: 菱田美紗紀）

第 12 章　中間発表の評価

本章では中間発表で記入してもらった評価シートの集計結果とコメントを参考にして今後の改善点を記述する。　評価シートの評価項目は「発表技術」と「発表内容」の 2 つと「発表内容」の細部に「ブラックジャックのルール説明の評価」、「検証の評価」の 2 つ合わせて計 4 つ用意した。そして、それぞれについて 1(非常に悪い) から 10（非常に優秀）までの間で評価点を付け、それぞれについてのコメント（評価理由）やアドバイスを記入する欄を用意した。

（※文責: 葛西隼人）

12.1　中間発表

12.1.1　評価点数の集計

中間発表で記入してもらった評価シートは計 42 枚だった。シートを記入した人の所属の分布の表 12.1 のようになった。

表 12.1　評価人数集計

所属	学年	人数
教員		6
一般		0
学生	院 2 年	0
学生	院 1 年	0
	学部 4 年	1
	学部 3 年	34
	学部 2 年	1
	学部 1 年	0
合計		42

評価人数の構成としては学部 3 年が大半を占めていた。その他には教員、学部 4 年と学部 1 年から 1 人ずつであった。次はそれぞれの評価項目についての平均点を表 12.2 に記す。

表 12.2　評価点数集計

所属	学年	発表技術	発表内容	ルール説明	検証説明
教員		8	7.5	7.83	7.67
学生		6.47	7.03	7.56	6.22
	学部 2,4 年	6	7	5.5	7
	学部 3 年	6.38	7.01	7.68	6.18
全体		6.69	7.1	7.6	6.43

全体の平均は「発表技術」については 6.69、「発表内容」については 7.1、「ルール説明」については 7.6、「検証説明」については 6.43 となった。それぞれの項目について高く評価したのは「教員」だった。次に、それぞれの結果を図 12.1、図 12.2、図 12.3、図 12.4 に示す。

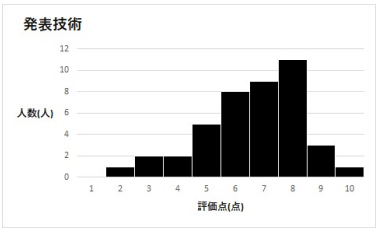


図 12.1　発表技術の評価グラフ

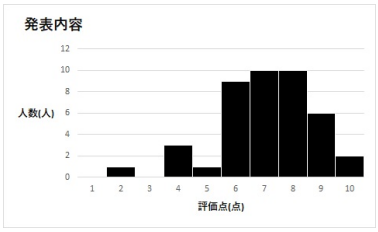


図 12.2　発表内容の評価グラフ

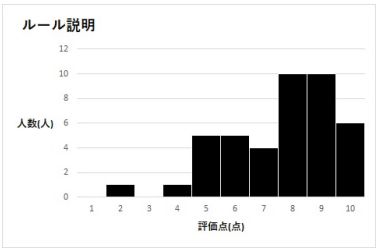


図 12.3　ルール説明の評価グラフ

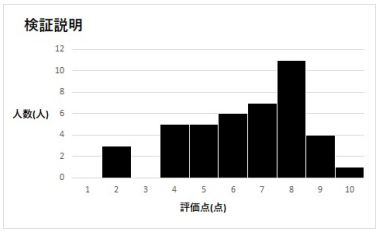


図 12.4　検証説明の評価グラフ

「発表技術」と「発表内容」については評価点が共に 7、8 点と高い評価点が多かった。ルール説明についても同様に高い評価点が多かった、一方で検証説明については 4、5 点が多い結果となった。

また、「発表技術」と「発表内容」の評点の相関係数は 0.71 となった。これは 2 つの評価項目がかなり関連してると言える数値である。次にコメントについて解析する。

（※文責: 葛西隼人）

12.1.2　コメント解析と改善点

まず、「発表技術」について、肯定的なコメントと否定的なコメントに分けて並べる。

肯定的なコメント

- 複雑な内容をうまく説明してくれた

Beat the Dealer! Mathematics of Complex Systems and Simulation.

- スライド内で図を多用されていて分かりやすかった

否定的なコメント

- 前を見て話してもらえないと聞き取りにくい
- 説明が少し早い
- プレゼンの内容が構造的でないために、全体に対して今どの部分を話しているのかわかりにくい

今回はスライドの分量が多く、どうしても早口になってしまったのでこのような意見が多かった。そしてスライドの完成が遅かったために構成までについて詳しく話し合うことができなかった。また発表の練習時間が取れなかったために技術面で足りない点もいくつかコメントで述べられていた。次に、「発表内容」についても同様に、コメントを肯定的なコメントと否定的なコメントに分けて並べる。肯定的なコメント

- 活動内容が明確でよかった
- 実験・検証が多く、説得力をもたせている
- 評価基準や比較対象が明確で理解しやすかった

否定的なコメント

- 用語についてより詳しく説明すべき
- 聞く人の知識が必要になるのでもう少しわかりやすく
- 検証結果のみせ方にもう少し工夫があると良かった。(専門的な知識がない人にもわかりやすく)

「発表内容」に関しては、ブラックジャックのルール説明はわかりやすいとのコメントが多かった。しかし検証の説明はあまり理解出来ないとのコメントも見受けられた。最後に「発表内容」と「発表技術」の 2 つのコメント欄から重要なアドバイスがいくつかあったため、これらについても述べていく。

- なぜあのような式を利用することで性能が表されているかの説明がもう少し欲しかった
- 文字数を減らし簡潔な内容の方がいいと思う
- スライドに色を使って見やすくしたほうが良い
- 基本的に文字がたくさんのスライドで、太字や下線などもなかったので、どこに注目してよいかわからなかった。後期の発表ではもっとスライドを効果的に見せてほしい

これらの意見に関しても、後期の活動に反映することとする。以上より中間発表の評価コメントは賛否両論であり、とても参考になった。

（※文責: 葛西隼人）

第 13 章　最終成果発表の評価

本章では最終成果発表で記入してもらった評価シートの集計結果とコメントを参考にして今後の改善点を記述する。　評価シートの評価項目は「発表技術」と「発表内容」の 2 つと「発表内容」の細部に「複雑性にちての説明の評価」、「カウンティングについての説明の評価」の 2 つ合わせて計 4 つ用意した。そして、それぞれについて 1(非常に悪い) から 10（非常に優秀）までの間で評価点を記入する欄、それぞれについてのコメント（評価理由）やアドバイスを記入する欄をそれぞれ用意した。

（※文責: 葛西隼人）

13.1　最終成果発表

13.1.1　評価点数の集計

最終成果発表で記入してもらった評価シートは計 50 枚だった。シートを記入した人の所属の分布は表 13.1 のようになった。

表 13.1　評価人数集計

所属	学年	人数
教員		7
一般		8
学生	院 2 年	0
学生	院 1 年	0
	学部 4 年	0
	学部 3 年	31
	学部 2 年	2
	学部 1 年	2
合計		50

中間発表では一般の評価人数が 0 人だったが、今回は函館市内の高校生が来ていたため一般の評価人数は増えて 8 人だった。学生と教員に関しては学部 4 年が 0 人となった代わりに、学部 1 年が 0 人となっていた点以外は中間発表とほとんど差異のない構成人数であった。次はそれぞれの評価項目についての平均点を表 13.2 に記す。

表 13.2　評価点数集計

所属	学年	発表技術	発表内容	複雑性説明	カウンティング説明
教員		7.42	7.14	7.14	6.71
一般		7.38	7	6.12	6.75
学生		7	7.2	6.66	7.02
	学部 1,2 年	6.5	6.25	6.5	6.5
	学部 3 年	6.87	7.39	6.7	7.26
全体		7	7.2	6.66	7.02

全体の平均は「発表技術」については 7 点、「発表内容」については 7.2 点、「複雑性説明」については 6.66 点、「カウンティング説明」については 7.02 点となった。中間発表と最終成果発表で「発表技術」のみ比較を行う。中間発表では 6.69 点、最終成果発表では 7 点となった。点数のみの比較でも 0.31 点高くなっている。これは中間発表からの技術改善が見られていると考えられる。「発表内容」については中間発表と異なるので比較は行わない。次に、それぞれの結果を図 13.1、図 13.2、図 13.3、図 13.4 に示す。

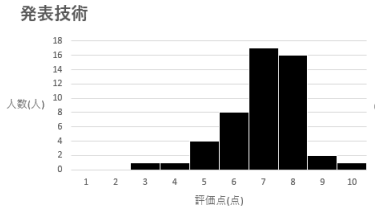


図 13.1　発表技術の評価グラフ

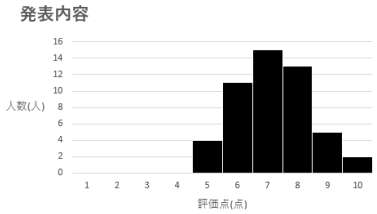


図 13.2　発表内容の評価グラフ

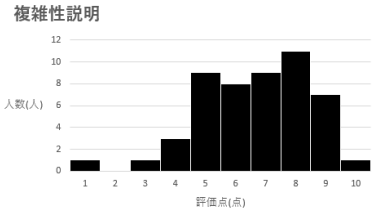


図 13.3　複雑性説明の評価グラフ

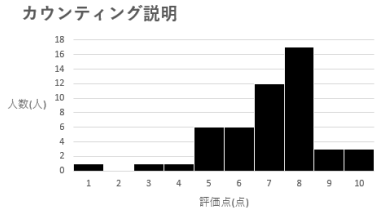


図 13.4　カウンティング説明の評価グラフ

「発表技術」と「発表内容」については評価点が共に 7,8 点と高い評価点が多かった。そして、「発表内容」に関しては最も低くても 5 点と全体的には評価はまとまっていたが、他のグラフは 1,3 点など低い点数があり全体的な評価は散らばっていた結果となった。「複雑性説明」では 5〜8 点の間にヒストグラムが集中していて理解度に差あることがわかった。

また、「発表技術」と「発表内容」の評価点数の相関係数は 0.73 となった。これは強い正の相関があり、2 つの評価項目がかなり関連してると言える数値である。次にコメントについて解析する。

（※文責: 葛西隼人）

13.1.2　コメント解析と改善点

まず、「発表技術」について、肯定的なコメントと否定的なコメントに分けて並べる。
肯定的なコメント

- アニメーションがついてて見やすいスライドだと思った
- アジェンダがあり、長いプレゼンがわかりやすかった
- スライドの内容はよく整理されておりわかりやすかったです

否定的なコメント

- 発表場所が悪かったのもあり声がきこえないところがあった
- 発表者が原稿を読んでいる感じがしたのが残念だった
- 発表が単調で何が大事なポイントが分かりづらかった

スライド構成については評価するコメントはあったが、声が小さい、スライドの文字ばかり見ているとの発表者の技術面についての否定的な意見が多かった。そして発表の時間配分が他のプロジェクトと異なっていたため発表を全て聞けなかった人が少しいた。次に、「発表内容」についても同様に、肯定的なコメントと否定的なコメントに分けて並べる。
肯定的なコメント

- 仮説・実験・検証が適切に行われていた
- 実験・検証が多く、説得力をもたせている
- プロジェクト目標の設定が明確であった
- 人間が使うことも含めた戦略の考察が面白いと思った

否定的なコメント

- エラー率は単純に複雑性から出すのではなく、似た状況がある時エラーするほうが正確なデータが出そうだと思った。
- ポスターにもスライドにもチームとしてどう取り組んだのか全く示されていないのは、プロジェクト学習の発表として不十分に思えます

そして要望としてのあったコメントをいくつかあったので記述する。

- 勝率を 100 パーセントにしてほしい
- 他のプロジェクトの発表もあるので、時間配分は考えてほしい

中間発表では、用語の説明が足りないというコメントが多く見受けられたが、最終成果発表ではカウンティングと複雑性の両方でわかりやすいというコメントがあった。今回の発表を通して得られた発表での留意点を以下に記述する。

- スライドは色を用いて見やすくする

Beat the Dealer! Mathematics of Complex Systems and Simulation.

- 太線や下線などの重要な箇所を視覚的に理解できるようにする
- 発表者は声は大きく、はっきり聞き取りやすく来ている人に目を向ける
- 用語の説明を入れる、構成はアジェンダを用いてわかりやすくする

本プロジェクトでの中間発表と最終成果発表は今後の研究活動にも参考になる点が数多くあり非常に有益なものであった。

（※文責: 葛西隼人）

参考文献

[1] Baldwin,R. and W. Cantey and H. Maisel and J. McDermott(1956) "The Optimum Strategy in Blackjack", *Journal of the American Statistical Association*,vol.51, no.275, 419-439.

[2] Chaitin,G.J. (1969)"On the Simplicity and Speed of Programs for Computing Infinite Sets of Natural Numbers", *Journal of the Association for Computing Machinery*, vol.16, no.3, 407-422.

[3] Jensen, K. (2014)"The Expected Value of an Advantage Blackjack player", *All Graduate Plan B and other Reports*, 524.

[4] Kelly,J,L. (1956) "A New Interpretation of Infotnation Rate", *THE BELL SYSTEM TECHNICAL JOURNAL*,vol35 , no.4, 917-926

[5] Miller,G.A. (1956)"The Magical Number Seven, Plus or Minus TwoSome Limits on Our Capacity for Processing Information", *American Psychological Association*, vol. 101, No. 2, 343-352.

[6] Python Software Foundation(2018), "Python 3.6.5 ドキュメント", <https://docs.python.jp/3/index.html> 2018 年 7 月 1 日アクセス.

[7] Python Software Foundation(2018) "9.6. random - 擬似乱数の生成" <https://docs.python.jp/3/ library/random.html> 2018 年 6 月 20 日アクセス

[8] Thorp,E.(1962) *Beat The Dealer:A Winning Strategy for the Game of Twenty One*,Vintage (宮崎三瑛 (2006)『ディーラーをやっつけろ!』, パンローリング)

[9] Vancura,O. and Fuchs,K.(1998) *KNOCK-OUT BLACKJACK, HUNTINGTON PRESS* (ライアン・モリス, 谷崎涼子 (2012) 『カードカウンティング入門 - カジノでたのしむブラックジャックテクニック』, パンローリング)

[10] 青木繁伸 (2010) "比率の差の多重比較（pairwise.prop.test の拡張）" <http://aoki2.si.gunma-u.ac.jp/R/p_multi_comp2.html> 2018 年 7 月 21 日アクセス

[11] 北野宏明 (1993)『遺伝的アルゴリズム』産業図書

[12] 斎藤康毅 (2016) 『ゼロから作る Deep Learning Python で学ぶディープラーニングの理論と実装』森北出版株式会社

[13] 斎藤隆浩 (1999) 『新訂ブラックジャック必勝法』株式会社データハウス

[14] 佐藤浩, 小野功, 小林重信 (1997) 『遺伝的アルゴリズムにおける世代交代モデルの提案と評価』, 人工知能学会誌,12,5,pp.734-744

[15] 全人類がわかる統計学 (2017) "カイ二乗検定を残差分析で評価する方法" <https://toukei.net/hypothesis-testing/chi2-test-residual-analysis/> 2018 年 7 月 6 日アクセス

[16] 竹澤邦夫 (2012) "19. 行列の作成" <http://cse.naro.affrc.go.jp/takezawa/r-tips/r/19.html> 2018 年 6 月 20 日アクセス

Beat the Dealer! Mathematics of Complex Systems and Simulation.

[17] 萩原将文 (1994) 『ニューロ・ファジィ・遺伝的アルゴリズム』産業図書株式会社

[18] 広井誠 (2007)”Algorithms with Python - 番外編:擬似乱数の検定” <http://www.geocities.jp/m_hiroi/light/pystat04.html> 2018 年 6 月 20 日アクセス

[19] 松本眞 (2013) ”Mersenne Twister Home Page” <http://www.math.sci.hiroshima-u.ac.jp/ m-mat/MT/what-is-mt.html> 2018 年 6 月 20 日アクセス

[20] 山内光哉 (1987) 『心理・教育のための統計法』株式会社サイエンス社

第 14 章　付録

Listing 14.1　card.py

```
class Card:
    RANKS = ('A', '2', '3', '4', '5', '6', '7', '8', '9', '10', 'J', 'Q', 'K')
    SUITS = ('Spade', 'Heart', 'Diamond', 'Club')

    def __init__(self, rank, suit):
        self.rank = rank
        self.suit = suit
        self.value = int(self.getvalue())

    # translate rank to number
    def getvalue(self):
        if self.rank == 'A':
            return 11
        elif self.rank == 'J' or self.rank == 'Q' or self.rank == 'K':
            return 10
        else:
            return self.rank
```

Listing 14.2　dealer.py

```
class Dealer(GamePlayer):
    def __init__(self, deckNum):
        self.deck = Deck(deckNum)
        self.totaldealerhandlist = [0] * 6
        self.shufflenum = 10000
        self.deck.shuffle(deckNum * self.shufflenum)
        # running count
        self.IRC = 0
        super().__init__()

    # to give out card
    def dealcard(self):

        # Assumption infinity deck
        card = Card(Card.RANKS[random.randrange(13)], "spade")
        return card
        """
        # Assumption limited deck
        # HiLow
        card = self.deck.Cards[self.deck.current]
        if 2 <= card.value <= 6:
            self.IRC = self.IRC + 1
        elif 10 <= card.value <= 11:
            self.IRC = self.IRC - 1

        self.deck.current += 1
        if self.deck.current == len(self.deck.Cards):
            self.deck.current = 0
        return card
        """


    # initial give out card
    def firstdeal(self, player):
        super().__init__()
        for x in player:
            x.initialize()
        firstdeal = 2
        while firstdeal > 0:
            self.cards.append(self.dealcard())
            for x in player:
                x.cards.append(self.dealcard())
            firstdeal -= 1

    # process until sum of hand be upper 17
    def continuehit(self):
        self.totalvalue()
        while (self.total < 17):
```

```
            self.cards.append(self.dealcard())
            self.totalvalue()
```

Listing 14.3 gamemanager.py

```
class GameManager:
    def __init__(self, players, dealer):
        self.players = players
        self.dealer = dealer
        self.checkdeal = True

    # judge dealer and each player
    def judge(self):
        for x in self.players:
            self.checkblackjack(x)
        self.checkblackjack(self.dealer)
        for player in self.players:
            if not player.surrendeflg:
                if player.burst == True:
                    if player.tag == "clone":
                        for i, x in enumerate(self.players):
                            if x.name == player.name:
                                self.players[i].addtotallose(player.betMoney)
                                break
                    player.addtotallose(player.betMoney)

                elif player.burst == False and self.dealer.burst == True:
                    # flag that player splits
                    spflg = False
                    for x in self.players:
                        if x.tag == "clone":
                            spflg = True

                    if player.tag == "clone":
                        for i, x in enumerate(self.players):
                            if x.name == player.name:
                                if player.naturalbj and not spflg:
                                    self.players[i].addtotalwin(player.betMoney*1.5)
                                    break
                                else:
                                    self.players[i].addtotalwin(player.betMoney)
                                    break
                    if player.naturalbj and not spflg:
                        player.addtotalwin(player.betMoney*1.5)
                    else:
                        player.addtotalwin(player.betMoney)

                elif player.total > self.dealer.total:
                    spflg = False
                    for x in self.players:
                        if player.tag=="clone":
                            spflg = True

                    if player.tag == "clone":
                        for i, x in enumerate(self.players):
                            if x.name == player.name:
                                if player.naturalbj and not spflg:
                                    self.players[i].addtotalwin(player.betMoney*1.5)
                                    break
                                else:
                                    self.players[i].addtotalwin(player.betMoney)
                                    break
                    if player.naturalbj and not spflg:
                        player.addtotalwin(player.betMoney*1.5)
                    else:
                        player.addtotalwin(player.betMoney)

                elif player.total < self.dealer.total:
                    if player.tag == "clone":
                        for i, x in enumerate(self.players):
                            if x.name == player.name:
                                self.players[i].addtotallose(player.betMoney)
                                break
                    player.addtotallose(player.betMoney)

                elif player.total == self.dealer.total:
                    if player.naturalbj and self.dealer.naturalbj:
                        if player.tag == "clone":
                            for i, x in enumerate(self.players):
                                if x.name == player.name:
                                    self.players[i].addtotaldraw()
```

```
                                    break
                            player.addtotaldraw()
                        elif player.naturalbj and self.dealer.normalbj:
                            if player.tag == "clone":
                                for i, x in enumerate(self.players):
                                    if x.name == player.name:
                                        self.players[i].addtotalwin(player.betMoney * 1.5)
                                        break
                            player.addtotalwin(player.betMoney * 1.5)
                        elif player.normalbj and self.dealer.naturalbj:
                            if player.tag == "clone":
                                for i, x in enumerate(self.players):
                                    if x.name == player.name:
                                        self.players[i].addtotallose(player.betMoney)
                                        break
                            player.addtotallose(player.betMoney)
                        elif player.normalbj and self.dealer.normalbj:
                            if player.tag == "clone":
                                for i, x in enumerate(self.players):
                                    if x.name == player.name:
                                        self.players[i].addtotaldraw()
                                        break
                            player.addtotaldraw()
                        else:
                            if player.tag == "clone":
                                for i, x in enumerate(self.players):
                                    if x.name == player.name:
                                        self.players[i].addtotaldraw()
                                        break
                            player.addtotaldraw()

    # discriminate natural or normal blackjack
    # given player or dealer as input
    def checkblackjack(self, player):
        if player.total == 21:
            if len(player.cards) == 2:
                player.naturalbj = True
            else:
                player.normalbj = True
```

Listing 14.4 gameplayer.py

```
class GamePlayer:

    def __init__(self):
        self.cards = []
        self.total = 0
        self.acetotal = 0
        self.usedace = 0
        self.burst = False
        self.naturalbj = False
        self.normalbj = False
        self.surrendeflg = False

    def initialize(self):
        self.cards = []
        self.total = 0
        self.acetotal = 0
        self.usedace = 0
        self.burst = False
        self.naturalbj = False
        self.normalbj = False
        self.surrendeflg = False

    # This function returns sum of player hand
    def totalvalue(self):
        i = 0
        self.total = 0
        self.acetotal = 0
        cardnum = len(self.cards)

        while i < cardnum:
            if self.cards[i].rank == 'A':
                self.acetotal += 1
            self.total += self.cards[i].value
            i += 1
        self.total -= 10 * self.usedace

        # jundge bust of player
        if self.total > 21:
            if self.acetotal - self.usedace > 0:
```

```
                self.total -= 10
                self.usedace += 1
                if self.total > 21:
                    self.burst = True
            else:
                self.burst = True
```

Listing 14.5 main.py

```
def main(strategy):
    # create player instance
    p1 = Player("player1")

    # add player who joins game
    players = [p1]

    # create dealer instance
    # the argument expresses deck num used in game
    dealer = Dealer(6)

    # this variable expresses cutcard
    cutcard = len(dealer.deck.Cards) / 2

    totalGameNum = 40
    remainingGameNum = 0

    minbet = 10

    maxbet = 10000

    trial = 50000
    trialNum = trial

    split_strategy = [["H", "H", "P", "P", "P", "P", "H", "H", "H", "H"],  # 2,2
                      ["P", "P", "P", "P", "P", "P", "H", "H", "H", "H"],  # 3,3
                      ["H", "H", "H", "P", "P", "H", "H", "H", "H", "H"],  # 4,4
                      ["D", "D", "D", "D", "D", "D", "D", "D", "H", "H"],  # 5,5
                      ["P", "P", "P", "P", "P", "H", "H", "H", "H", "H"],  # 6,6
                      ["P", "P", "P", "P", "P", "P", "H", "H", "H", "H"],  # 7,7
                      ["P", "P", "P", "P", "P", "P", "P", "P", "P", "P"],  # 8,8
                      ["P", "P", "P", "P", "P", "S", "P", "P", "S", "S"],  # 9,9
                      ["S", "S", "S", "S", "S", "S", "S", "S", "S", "S"],  # 10,10
                      ["P", "P", "P", "P", "P", "P", "P", "P", "P", "P"]  # A,A
                      ]

    while True:
        # main loop
        remainingGameNum = totalGameNum
        dealer.deck.shuffle(dealer.shufflenum)
        players[0].money = 1000
        dealer.IRC = 0
        if trialNum \% 100 == 0:
            print(trialNum)

        while True:
            if remainingGameNum \% 100 == 0:
                print(remainingGameNum)

            # ゲームを始める前にデッキの中からカットカードが出てきているかを確認し、出てきていれば、デッキをシャッフルする
            """
            # 有限デックの際のシャッフルを行うタイミングの記述
            if dealer.deck.current > cutcard:
                dealer.deck.shuffle(dealer.shufflenum)
                dealer.IRC = -20
            """

            # ランニングカウントによって賭け額を変化させる
            # 今回はHigh-法を適用しているLow
            rc = 1
            if dealer.IRC <= 1:
                rc = 1
            else:
                rc = dealer.IRC

            if players[0].money == 0:
                money_list.append(None)
                card_num_list.append(None)

                # ループの処理
                remainingGameNum -= 1
```

```
72                  if remainingGameNum <= 0:
73                      break
74
75                  # 各プレイヤーのベット
76                  for player in players:
77                      if player.money < minbet * rc:
78                          player.bet(player.money)
79                      else:
80                          player.bet(minbet * rc)
81
82                  # 参加プレイヤーの初期化を実行後、ディーラーが各プレイヤー（自身含む）に初期カードを配る
83                  dealer.firstdeal(players)
84
85                  # 各プレイヤーの点数を更新
86                  for player in players:
87                      player.totalvalue()
88
89                  # サレンダーの回数を記録する
90                  surrenderCounter = 0
91
92                  # スプリットの回数とダブルダウンの回数を記録する
93                  spCounter = 0
94                  ddCounter = 0
95
96                  # スプリットするかどうかを確認する
97                  for i, player in enumerate(players):
98                      if player.cards[0].rank == player.cards[1].rank:
99                          usermessage = split_strategy[player.cards[0].value - 2][dealer.
                              cards[0].value - 2]
100                         if (usermessage == 'P' or usermessage == 'p') and player.money > 2
                              * player.betMoney:
101                             # プレイヤーのクローンを作成し、ゲームに参加するプレイヤーとして追加登録する
102                             playerClone = Player(player.name,money=player.money - player.
                                  betMoney, betMoney=player.betMoney, tag="clone")
103                             players.insert(i + 1, playerClone)
104                             spCounter += 1
105
106                             # クローンにプレイヤーが所持しているカードを一枚渡す
107                             playerClone.dealedcard(player.cards[1])
108                             del player.cards[1]
109
110                             # 使用済みの数を初期化するA
111                             player.usedace = 0
112                             playerClone.usedace = 0
113
114                             # プレイヤーとクローンにカードを配り直す
115                             player.dealedcard(dealer.dealcard())
116                             playerClone.dealedcard(dealer.dealcard())
117
118                 for player in players:
119                     if player.tag == "clone":
120                         players[0].totalsplit += 1
121
122
123                 # 各プレイヤーに対して選択肢を提示する
124                 for player in players:
125                     while True:
126                         # プレイヤーの選択はベーシックストラテジーに沿って行われるものとする
127                         # プレイヤーの手札にA(11)が残っている場合
128                         if player.acetotal - player.usedace > 0:
129                             usermessage = strategy[player.total + 6][dealer.cards[0].value
                                  - 2]
130                         # プレイヤーの手札にA(11)が残っていない場合
131                         else:
132                             usermessage = strategy[player.total - 4][dealer.cards[0].value
                                  - 2]
133
134                         # 一定条件下でプレイヤーが戦略を間違える際の処理
135
136
137                         # プレイヤーの選択による行動の分岐を記述
138                         # プレイヤーがヒットを選択した場合
139                         if usermessage == 'H' or usermessage == 'h':
140                             player.hit(dealer)
141                             if player.burst:
142                                 break
143
144                         # プレイヤーがスタンドを選択した場合
145                         elif usermessage == 'S' or usermessage == 's':
146                             player.stand()
147                             break
148
149                         # プレイヤーがダブルダウンを選択した場合
```

```
                        elif usermessage == 'D' or usermessage == 'd':
                            if players[0].money -spCounter*players[0].betMoney - ddCounter*
                              players[0].betMoney > player.betMoney * 2:
                                # ヒット後にはダブルダウンの選択不可
                                if len(player.cards) == 2:
                                    player.doubledown(dealer)
                                    ddCounter += 1
                                    break
                                else:
                                    player.hit(dealer)
                                    if player.burst:
                                        break
                            else:
                                player.hit(dealer)
                                if player.burst:
                                    break

                        # プレイヤーがサレンダーを選択した場合
                        elif usermessage == "R" or usermessage == "r":
                            # ヒット後にはサレンダーの選択不可
                            if len(player.cards) == 2:
                                player.surrender()
                                if player.tag == "clone":
                                    players[0].totalsurrender += 1
                                surrenderCounter += 1
                                break
                            else:
                                player.hit(dealer)
                                if player.burst:
                                    break

                # ディーラーはを超えるまでヒットを続ける17
                dealer.continuehit()

                # の初期化GameManager
                gamemanager: GameManager = GameManager(players, dealer)

                # 勝敗を判定する
                gamemanager.judge()

                # クローンを削除する
                while True:
                    cloneflg = False
                    for i, player in enumerate(players):
                        if player.tag == "clone":
                            del player
                            del players[i]
                            cloneflg = True
                    if not cloneflg:
                        break

                # ループの処理
                remainingGameNum -= 1
                if remainingGameNum <= 0:
                    break
        trialNum -= 1
```

Listing 14.6　player.py

```
class Player(GamePlayer):
    # プレイヤーの初期化
    def __init__(self, name, money=0, betMoney=0, tag="player"):
        # プレイヤー名
        self.name = name
        # 所持金
        self.money = money
        # ベット額クローンに値を渡す際に使用する()
        self.betMoney = betMoney
        # 累計勝利回数、敗北回数、引き分け回数
        self.totalwin = 0
        self.totallose = 0
        self.totaldraw = 0
        self.totalsplit = 0
        self.totalsurrender = 0
        self.totalplayerhandlist = [0] * 12
        # 勝利したか負けたかの確認
        self.winlose = ""
        # プレイヤーとクローンを見分ける
        self.tag = tag
        self.debagtxt = ""
        super().__init__()
```

```
23
24      # プレイヤーにカードを配るときに使用する関数
25      def dealedcard(self, card):
26          self.cards.append(card)
27          self.totalvalue()
28
29      # プレイヤー側のヒットの処理
30      def hit(self, dealer):
31          self.dealedcard(dealer.dealcard())
32          self.debagtxt += "H"
33
34      # プレイヤー側のスタンドの処理
35      def stand(self):
36          self.debagtxt += "S"
37          pass
38
39      # プレイヤー側のダブルダウンの処理
40      def doubledown(self, dealer):
41          self.debagtxt += "D("
42          self.betMoney *= 2
43          self.hit(dealer)
44          self.stand()
45          self.debagtxt += ")"
46
47      # サレンダーの処理
48      def surrender(self):
49          self.debagtxt += "R"
50          self.totalsurrender += 1
51          # self.surrenderflg = True
52          self.money -= self.betMoney/2
53
54      # プレイヤー側のベットの処理
55      def bet(self, betMoney):
56          self.betMoney = betMoney
57
58      # プレイヤーのインシュランスの処理
59      def insurance(self, dealer):
60          if dealer.cards[0] + dealer.cards[1] == 21:
61              self.money += self.betMoney
62          else:
63              self.money -= self.betMoney/2
64
65      # 自身の手札を表示するUI
66      def showhands(self):
67          for x in self.cards:
68               print('/', x.suit, x.rank)
69          print("---total---:␣", self.total, "\n")
70
71      # プレイヤーの勝利回数を増やす
72      def addtotalwin(self, money):
73          self.money += money
74          self.totalwin += 1
75          self.winlose = "win"
76
77      # プレイヤーの敗北回数を増やす
78      def addtotallose(self, money):
79          self.money -= money
80          self.totallose += 1
81          self.winlose = "lose"
82
83      # プレイヤーの引き分け回数を増やす
84      def addtotaldraw(self):
85          self.totaldraw += 1
86          self.winlose = "draw"
```

Listing 14.7　geneticalgorithm.py

```
1  import random
2  import numpy as np
3  import pickle
4  from multiprocessing import Pool
5
6  class GA:
7    def __init__(self, Np, Ng, Pc, Pm, locus_num):
8      self.Np = Np    #選出個体数(population number)
9      self.Ng = Ng    #最大世代(generation number)
10     self.Pc = Pc    #交叉率(probability of clossover)
11     self.Pm = Pm    #突然変異率(probability of mutation)
12     self.locus_num = locus_num   #遺伝子座
13
14   def Solve(self, func):
```

```
    group_hist = list()
    self.fitnessfunc = func
    p = Pool(10)

    #最大世代数分計算
    for i in range(self.Ng):
      #個体ごとの適応度を計算
      evaluation = p.map(self.GetFitness, self.group)
      if i > 0:
        #エリート個体と最低評価個体の入れ替え
        low_i = np.argmin(evaluation)
        self.group[low_i] = elite
        evaluation[low_i] = elite_score
      #履歴の保存
      if i % 10 == 0:
        with open('progress{}.dat'.format(int(i/10)), 'wb') as f:
          savedata = list(zip(self.group, evaluation))
          pickle.dump(savedata,f)
      group_hist.append(self.group)
      #適応度比例選択によって親個体を選出
      fitness = [e/sum(evaluation) for e in evaluation]
      candidate = [self.Selection(self.group, fitness) for j in range(self.Np)]
      #エリート優秀個体()の選出
      elite_i = np.argmax(evaluation)
      elite = list(self.group[elite_i])
      elite_score = evaluation[elite_i]
      #新世代の個体群
      newgroup = list()
      #親個体の選出
      while len(newgroup) != self.Np:
        #交叉のための親を選定
        parent1 = self.Selection(self.group, fitness)
        parent2 = self.Selection(self.group, fitness)
        #交叉もしくはコピーを作成
        children = list()
        if random.random() < self.Pc:
          children = self.UniformCrossOver(parent1, parent2)
        else:
          children.append(parent1)
          children.append(parent2)
        #突然変異を適用
        for child in children:
          self.Mutation(child)
          newgroup.append(child)
      self.group = list(newgroup)

    with open('garesult.dat', 'wb') as f:
      pickle.dump(group_hist, f)

  def GetFitness(self, individual):
    fitness = self.fitnessfunc(individual)
    return fitness

  def Mutation(self, individual):
    for i in range(self.locus_num):
      if random.random() < self.Pm:
        individual[i] = '1' if individual[i] == '0' else '0'

  def CrossOver(self, parent1, parent2):
    return self.UniformCrossOver(parent1, parent2)

  def UniformCrossOver(self, parent1, parent2):
    child1 = list()
    child2 = list()
    mask = [0 if random.random() < 0.5 else 1 for i in range(self.locus_num)]
    for i,mb in enumerate(mask):
      if mb == 1:
        child1.append(parent2[i])
        child2.append(parent1[i])
      else:
        child1.append(parent1[i])
        child2.append(parent2[i])
    return [child1, child2]

  def Selection(self, group, fitness):
    return self.FitnessProportionalSelection(group, fitness)

  def FitnessProportionalSelection(self, group, fitness):
    choiced_index = np.random.choice([i for i in range(len(group))], p=fitness)
    candidate = group[choiced_index]
    return candidate

  def GenerateGroup(self, individual_num):
```

```
98      self.group = [self.GenerateIndividual() for i in range(individual_num)]
99      return
100
101   def GenerateIndividual(self):
102     individual = list()
103     for i in range(self.locus_num):
104       num_str = '0' if random.random() < 0.5 else '1'
105       individual.append(num_str)
106     return individual
```

Listing 14.8 searchbyga.py

```
1   import numpy as np
2   import sys
3   import argparse
4   import json
5   from geneticalgorithm import GA
6   import BlackJackSimulator
7   import pprint
8
9   def simulate(individual):
10    score = 1
11    #戦略表の作成
12    strategy = ['0' for i in range(8*10)] #hit from 4 to 11,hard hand
13    strategy.extend([individual[i] for i in range(9*10)])
14    strategy.extend(['1' for i in range(10)])
15    strategy.extend([individual[i+90] for i in range(9*10)])
16    strategy.extend(['1' for i in range(10)])
17    strategy = list(np.array(list(map(lambda x: 'H' if x == '0' else 'S', strategy))).reshape
        (28, 10))
18    #ゲームのシミュレート
19    config = dict()
20    config['maxbet'] = 1
21    config['minbet'] = 1
22    config['gamenum'] = 50000
23    config['decknum'] = 1
24    config['exportresult'] = False
25    config['exportdebug'] = False
26
27    result = BlackJackSimulator.main(strategy, config)
28    totalwin = result[0]['totalwin']
29
30    #連長圧縮個体の遺伝子のみから計算()
31    count = 0
32    for row_i in range(18):
33      before = 'B'
34      for col_i in range(10):
35        if not (individual[row_i*10+col_i] == before):
36          count += 2
37        before = individual[row_i*10+col_i]
38
39    win_rate = totalwin / 50000
40    complexity = count / 180
41
42    score = win_rate - complexity + 10
43
44    return score
45
46  if __name__ == '__main__':
47    #コマンドライン引数についての設定
48    parser = argparse.ArgumentParser(
49                prog="searchbyga",
50                usage="python␣searchbyga␣[simulator␣config␣file]",
51                description="",
52                epilog="",
53                add_help=True
54              )
55    #に関する設定ファイルを指定する引数GA
56    parser.add_argument(
57        "simulatorconfigfile",
58        help="The␣json␣file␣what␣configures␣ga␣simulation."
59    )
60
61    args = parser.parse_args()
62
63    #シミュレーションの設定ファイルを読み込む
64    with open(args.simulatorconfigfile, "r") as f:
65      datadic = json.load(f)
66      Np = datadic['population_number']
67      Ng = datadic['generation_number']
68      Pc = datadic['probability_crossover']
```

```
    Pm = datadic['probability_mutation']
    locus_num = datadic['locus_num']

  ga = GA(Np, Ng, Pc, Pm, locus_num)
  ga.GenerateGroup(ga.Np)

  for i in range(5):
    basicstrategy = list()
    basicstrategy.extend(['h','h','s','s','s','h','h','h','h','h'])
    basicstrategy.extend(['s','s','s','s','s','h','h','h','h','h'])
    basicstrategy.extend(['s','s','s','s','s','h','h','h','h','h'])
    basicstrategy.extend(['s','s','s','s','s','h','h','h','h','h'])
    basicstrategy.extend(['s','s','s','s','s','h','h','h','h','h'])
    basicstrategy.extend(['s','s','s','s','s','s','s','s','s','s'])
    basicstrategy.extend(['s','s','s','s','s','s','s','s','s','s'])
    basicstrategy.extend(['s','s','s','s','s','s','s','s','s','s'])
    basicstrategy.extend(['s','s','s','s','s','s','s','s','s','s'])

    basicstrategy.extend(['h','h','h','h','h','h','h','h','h','h'])
    basicstrategy.extend(['h','h','h','h','h','h','h','h','h','h'])
    basicstrategy.extend(['h','h','h','h','h','h','h','h','h','h'])
    basicstrategy.extend(['h','h','h','h','h','h','h','h','h','h'])
    basicstrategy.extend(['h','h','h','h','h','h','h','h','h','h'])
    basicstrategy.extend(['h','h','h','h','h','h','h','h','h','h'])
    basicstrategy.extend(['S','S','S','S','S','S','S','h','h','S'])
    basicstrategy.extend(['S','S','S','S','S','S','S','S','S','S'])
    basicstrategy.extend(['S','S','S','S','S','S','S','S','S','S'])

    for j,x in enumerate(basicstrategy):
      basicstrategy[j] = '0' if x == 'H' or x == 'h' else '1'

    ga.group[i] = basicstrategy

  for i in range(5):
    strategy = list()
    strategy.extend(['h','h','h','h','h','h','h','h','h','h'])
    strategy.extend(['h','h','h','h','h','h','h','h','h','h'])
    strategy.extend(['s','s','s','s','s','s','s','s','s','s'])
    strategy.extend(['s','s','s','s','s','s','s','s','s','s'])
    strategy.extend(['s','s','s','s','s','s','s','s','s','s'])
    strategy.extend(['s','s','s','s','s','s','s','s','s','s'])
    strategy.extend(['s','s','s','s','s','s','s','s','s','s'])
    strategy.extend(['s','s','s','s','s','s','s','s','s','s'])
    strategy.extend(['s','s','s','s','s','s','s','s','s','s'])

    strategy.extend(['h','h','h','h','h','h','h','h','h','h'])
    strategy.extend(['h','h','h','h','h','h','h','h','h','h'])
    strategy.extend(['h','h','h','h','h','h','h','h','h','h'])
    strategy.extend(['h','h','h','h','h','h','h','h','h','h'])
    strategy.extend(['h','h','h','h','h','h','h','h','h','h'])
    strategy.extend(['h','h','h','h','h','h','h','h','h','h'])
    strategy.extend(['S','S','S','S','S','S','S','h','h','S'])
    strategy.extend(['S','S','S','S','S','S','S','S','S','S'])
    strategy.extend(['S','S','S','S','S','S','S','S','S','S'])

    for j,x in enumerate(strategy):
      strategy[j] = '0' if x == 'H' or x == 'h' else '1'

    ga.group[i+5] = strategy

  for i in range(5):
    strategy = list()
    strategy.extend(['h','h','h','h','h','h','h','h','h','h'])
    strategy.extend(['h','h','h','h','h','h','h','h','h','h'])
    strategy.extend(['h','h','h','h','h','h','h','h','h','h'])
    strategy.extend(['h','h','h','h','h','h','h','h','h','h'])
    strategy.extend(['h','h','h','h','h','h','h','h','h','h'])
    strategy.extend(['s','s','s','s','s','s','s','s','s','s'])
    strategy.extend(['s','s','s','s','s','s','s','s','s','s'])
    strategy.extend(['s','s','s','s','s','s','s','s','s','s'])
    strategy.extend(['s','s','s','s','s','s','s','s','s','s'])
    strategy.extend(['s','s','s','s','s','s','s','s','s','s'])

    strategy.extend(['h','h','h','h','h','h','h','h','h','h'])
    strategy.extend(['h','h','h','h','h','h','h','h','h','h'])
    strategy.extend(['h','h','h','h','h','h','h','h','h','h'])
    strategy.extend(['h','h','h','h','h','h','h','h','h','h'])
    strategy.extend(['h','h','h','h','h','h','h','h','h','h'])
    strategy.extend(['h','h','h','h','h','h','h','h','h','h'])
    strategy.extend(['S','S','S','S','S','S','S','h','h','S'])
    strategy.extend(['S','S','S','S','S','S','S','S','S','S'])
    strategy.extend(['S','S','S','S','S','S','S','S','S','S'])

    for j,x in enumerate(strategy):
```

```
      strategy[j] = '0' if x == 'H' or x == 'h' else '1'

    ga.group[i+10] = strategy

  for i in range(5):
    strategy = list()
    strategy.extend(['h','h','h','h','h','h','h','h','h','h'])
    strategy.extend(['h','h','h','h','h','h','h','h','h','h'])
    strategy.extend(['h','h','h','h','h','h','h','h','h','h'])
    strategy.extend(['s','s','s','s','s','s','s','s','s','s'])
    strategy.extend(['s','s','s','s','s','s','s','s','s','s'])
    strategy.extend(['s','s','s','s','s','s','s','s','s','s'])
    strategy.extend(['s','s','s','s','s','s','s','s','s','s'])
    strategy.extend(['s','s','s','s','s','s','s','s','s','s'])
    strategy.extend(['s','s','s','s','s','s','s','s','s','s'])

    strategy.extend(['h','h','h','h','h','h','h','h','h','h'])
    strategy.extend(['h','h','h','h','h','h','h','h','h','h'])
    strategy.extend(['h','h','h','h','h','h','h','h','h','h'])
    strategy.extend(['h','h','h','h','h','h','h','h','h','h'])
    strategy.extend(['h','h','h','h','h','h','h','h','h','h'])
    strategy.extend(['h','h','h','h','h','h','h','h','h','h'])
    strategy.extend(['S','S','S','S','S','S','S','h','h','S'])
    strategy.extend(['S','S','S','S','S','S','S','S','S','S'])
    strategy.extend(['S','S','S','S','S','S','S','S','S','S'])

    for j,x in enumerate(strategy):
      strategy[j] = '0' if x == 'H' or x == 'h' else '1'

    ga.group[i+15] = strategy

  ga.Solve(simulate)
```