卒業研究報告書

題目

ポケモンカードゲーム
デッキ作成補助プログラムの開発

指導教員

石水 隆 講師

報告者

16-1-037-0140

黒瀬 友梧

近畿大学理工学部情報学科

令和３年２月１日提出

概要

トレーディングカードゲームは,ルールに則りデッキを作って持ち寄り,2 人以上で行うゲームである.トレーディングカードゲームは近年市場規模及び売り上げを伸ばしており,スマートフォンやPC 向けアプリケーションのデジタルカードゲームも展開されている.

既存のデジタルカードゲームは,自動的にデッキを構築する機能や直前にどのカードがどう使われたか等をログとして表示する機能など,便利な機能やアプリケーションが存在していることが多い.これらの機能をトレーディングカードゲーム向けに作れないか考えた.

本研究では,ポケモンカードを題材としデッキ作成を補助するプログラムの作成を行う.

内容

1. 序論

1.1. 本研究の背景

トレーディングカードゲームとは,ルールに則り組み合わせたカードの束(デッキと呼ぶ)を持ち寄り,2 人以上で対戦を行うゲームである.1993 年に世界初のトレーディングカードゲームとしてアメリカで「マジック・ザ・ギャザリング」が発売したことを皮切りに,それを輸入した日本国内でもヒットした.現在トレーディングカードゲームは国内玩具市場の中でも規模が大きく,また近年では「Shadowverse」 [1]や「Hearthstone」 [2]といったスマートフォンや PC 向けに作られたデジタルカードゲームも存在している.

図 1 は Heathstone とそのゲーム向けに開発された HearthstoneDeckTracker [3]の動作例である.左側に相手プレイヤーが使ったカードや手札の枚数,残りデッキ枚数を表示している.また,右側に自分が使っているデッキが表示されており,デッキに残っていないカードは暗く表示している.

図 1Hearthstone 向けに開発された HearthstoneDeckTracker 実行例

このように,デジタルカードゲームは,対戦する媒体によって操作を円滑にするアプリケーションが存在している.こういったアプリケーションはデジタルだからこそ実現できる訳だが,アナログのカードゲーム向けにもこういったアプリケーションがあれば,アナログカードゲームにデジタルカードゲームのようなゲームの円滑性が得られる.いくつかのトレーディングカードゲームに対しては,デッキ作成をサポートするサービスがある. [4] [5]また,個人でデッキ作成に役立つサービスを提供している例もある. [6]

1.2. 本研究の目的

本研究では,ポケモンカードを対象にしたアプリケーションを開発する.アナログゲームであるため,デジタルカードゲームのように対戦中に動作するのは現実的ではない.そのため,本研究では一旦構築したデッキを再構築するのを補助するプログラムを作成する.

また,既存のサービスに理論値を算出するプログラムが存在する.しかし,イベント等で行われるゲーム数は多くて 30 ゲームであり,既存のサービスではその時に起こりうる偏りのあるデータを求めることができないため,このようなデータも集められるようにしたい.

1.3. 本報告書の構成

本報告書の構成を以下に示す.2 章でポケモンカード及びトレーディングカードゲームについて述べる.3 章で作成したプログラムについて説明し,4 章で検証と考察を行い,5 章で結論と今後の課題について述べる.

2. ポケモンカードゲーム

本章では,研究対象であるポケモンカード及びトレーディングカードゲームの用語等について述べる.

2.1. ポケモンカードゲームとは

ポケモンカードゲーム(以下ポケモンカード)とはゲーム「ポケットモンスター」シリーズ内でのポケモンバトルを再現したカードゲームである. [7]日本国産としては初の本格的トレーディングカードゲームであり,1996 年 10 月 20 日に最初の商品が販売され,累計出荷枚数は 304 億枚以上である.また,国外にも展開しており,現在では 13 言語のカードがあり,77 エリアで販売されている. [8]

2.2. トレーディングカードゲームとは

トレーディングカードゲーム(以下 TCG)とは,各プレイヤーが持ち込んだトレーディングカードからルールに則り組み合わせたデッキを持ち寄り,2 人以上で対戦を行うゲームである.トレーディングカードは英語圏ではコレクタブルカードとも呼ばれ,元々は収集目的であることが多かったが,トランプや UNO といったカードゲームの発展形とし考案され,ウィザーズ・オブ・ザ・コースト社が 1993 年に発売した「マジック・ザ・ギャザリング」 [9] [10]がトレーディングカードゲームの原点とされている.

2.2.1. デッキ

デッキとは,プレイヤーがカードゲームを行う上で用意する必要のあるカードの束である.このデッキを作る際には,各 TCG のルールに則る必要があり,原則複数の種類の TCG を合わせることはできない.表 1 に主要な TCG のデッキ作成ルールを示す.

表 1 主要 TCG のデッキ作成ルール

TCG 名	デッキ枚数	同名カードの枚数	その他
マジック・ザ・ギャザリング [11]	原則 60 枚以上	原則 4 枚まで	形式によって一部最大枚数が設定されている
ポケモンカードゲーム [12]	60 枚	原則 4 枚まで	形式によって禁止カードが設定されている たねポケモンは 1 枚以上いれなければいけない
遊戯王オフィシャルカードゲーム [13]	40 枚~60 枚	3 枚まで	リミットレギュレーションにより一部最大枚数が設定されている
デュエル・マスターズ [14]	40 枚	4 枚まで	殿堂レギュレーションにより一部最大枚数が設定されている

2.2.2. マリガン

マリガンとは,ゴルフ競技において「そのホールの第一打目をペナルティーなしで打ち直すこと」である.それに由来し,カードゲームでは「最初に配られた手札を特定の条件下で引き直すこと」とされている.

カードゲームの種類によってマリガンができるかどうか,またできる条件等が定められており,ポケモンカードでは「最初の手札にたねポケモンが一枚もなければ相手にないことを確認してもらった後引き直すこと」をマリガンと言われている.

2.2.3. サイド

ポケモンカードで最初に手札を引いてたねポケモンカードを置いた後に自分の場の左側に並べるカードのことである.デッキの枚数が 60 枚の場合 6 枚並べる.対戦相手のポケモンを倒すと,その目印として倒したポケモンの種類に応じて枚数とる.サイドを先に取り切ることはプレイヤーが勝利する条件の一つである.

2.2.4. カードゲームとコンピュータ

過去に株式会社コナミデジタルエンタテインメントは遊戯王オフィシャルカードゲーム [15]を題材としたコンシューマゲームを展開していた. [16]ゲームである以上ノンプレイヤーキャラクター(以下 NPC)を実装する必要があるが,NPC の強さに調整に苦悩しているよう思えた.

カードゲームは不完全情報ゲームであり,カードの種類も日々増加している.コンシューマゲームのような限られた容量の中で全てのカードを完璧に使う NPC を実装するのは非常に困難である.高難易度に設定されるデッキはある特定のカードを組み合わせるコンボデッキであることが多いため,必要以上に長考したり,必要なカードを意味もなく使ってしまったりすることがある.そのため,ゲームの表記上の強さと実際の強さがあべこべになる現象が起こってしまう.

しかし,カードの種類が比較的少なく,コンピュータで動作するデジタルカードゲームでは Ai を用いて強力な NPC を実装する事例がある. [17]そのため,チェスや将棋のように,将来人間より強いカードゲーム NPC が実現されると考えられる.

2.3. ポケモンカードのゲーム準備の流れ

本節ではポケモンカードのゲーム準備の流れを説明する.以下の手順に沿って準備を進める

1. デッキをよくシャッフルし,裏側にして置く
2. 置いたデッキからカードを 7 枚引き,たねポケモンがいるかを確認する
3. ない場合は相手にそれを確認しカードをデッキに戻し再び手順 1,2 を行う
   あった場合はそれを裏側でバトル場に置く
4. 置いたデッキからカードを 6 枚,表を見ず裏側でサイドに置く
5. お互いプレイヤーが 1 から 4 を完了したらゲーム開始

より詳しい内容は文献目録 [18]を参照である.

2.4. ポケモンカードの種類

ポケモンカードは大きく分けて「ポケモン」,「エネルギー」,「トレーナーズ」,「特別なカード」の 4 種類ある(図 2).

デッキにはこの 4 種類を組み合わせて作るが,「ポケモン」と「特別なカード」に存在する「たねポケモン」を最低 1 枚以上デッキにいれて,合計 60 枚にする必要がある.また,「エネルギー」に含まれる「基本エネルギー」には入れられる枚数に制限はなく,それ以外の種類のカードはレアリティ問わず同名合わせて最大で 4 枚までデッキにいれることができる. [19]

今回のプログラムでは「カードの名前」と「カードの種類」がわかればプログラム上でデッキを

作ることができるため，現物の用意は不要である．

図 2　ポケモンカードの種類 [19] [20]

3.　開発したプログラムについて

本章では，本研究で作成したデッキ作成補助プログラムについて記述する．付録に本研究で作成したプログラムのソースコードを示す．

本研究で作成したプログラムは，ゲームの準備を実際と同じように行い初手もしくはサイドに任意のカードがどれだけ存在するかを集計する．この集計は，人力でも行うことができるが，十分なデータ量を得るには非常に効率が悪い．また，理論値を求めるだけでは出力できない，数～数十試合を想定した際に起こりうる値の偏りも算出できる．これらを集計したデータを実際のポケモンカードのデッキ作成の参考にする．

本研究で作成したプログラムは，DeckAssist．java，SixtyDeckmaker．java，EternutsVMAX．java の 3 つから成る．

3．1．プログラムの仕様

プログラムを実行すると，初期設定ではデッキが生成され，0～6 のいずれかとカード名を入力することで，各メソッドの結果が出力される．各メソッドより詳しい出力結果は次節で説明する．

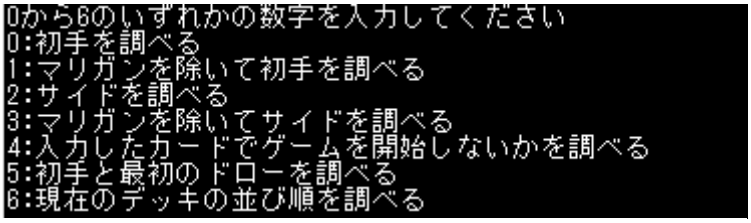

図 3 プログラムを実行した初期画面

3.2. DeckAssist.java

DeckAssist クラスは本研究で作成したプログラムの中心部分である.以下にこのクラスのクラス図を記載する.

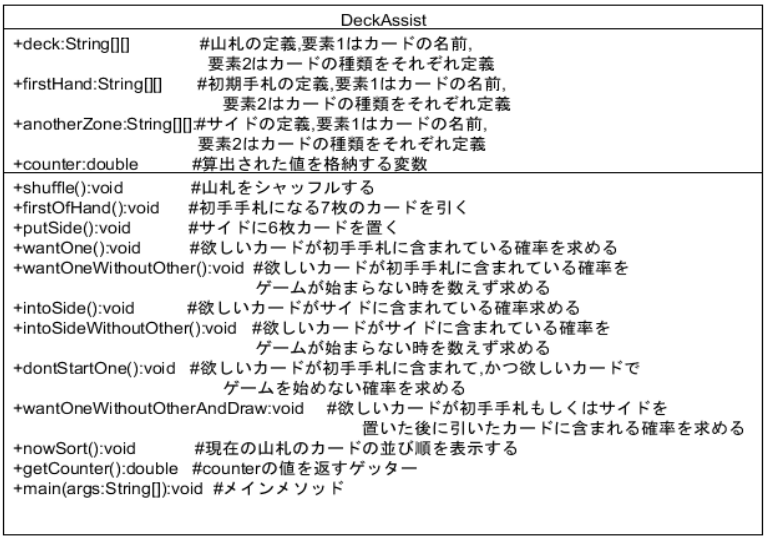

図 4DeckAssist のクラス図

3.2.1. shuffle メソッド

shuffle メソッドは,現在のデッキをシャッフルし順番を入れ替えるメソッドである.図 5 に DeckAssist を動作させた際の初期のデッキの並び順,図 6 に shuffle メソッドを実行後のデッキの並び順を記載している.

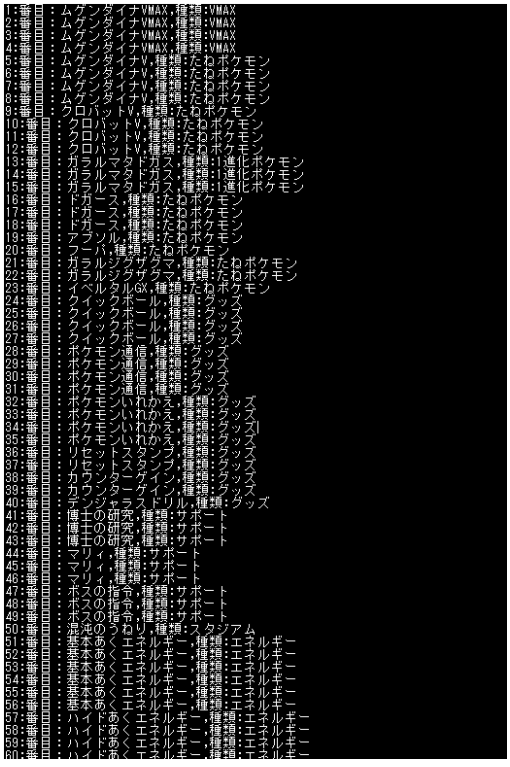

図 5 初期のデッキ並び順

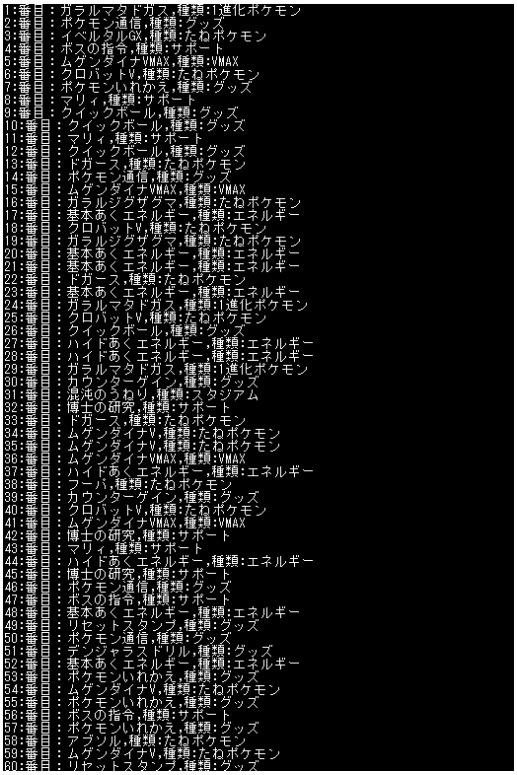

図 6shuffle メソッド実行後のデッキ並び順

3.2.2. fisrtOfSeven メソッド

firstOfSeven メソッドは初手となるカードをデッキの上から 7 枚引くメソッドである.ルールに則りシャッフルをしてからデッキの上を確認する.図 7 は初期の並び順を出力した後 fisrtOfSeven メソッドを実行した出力を示している.

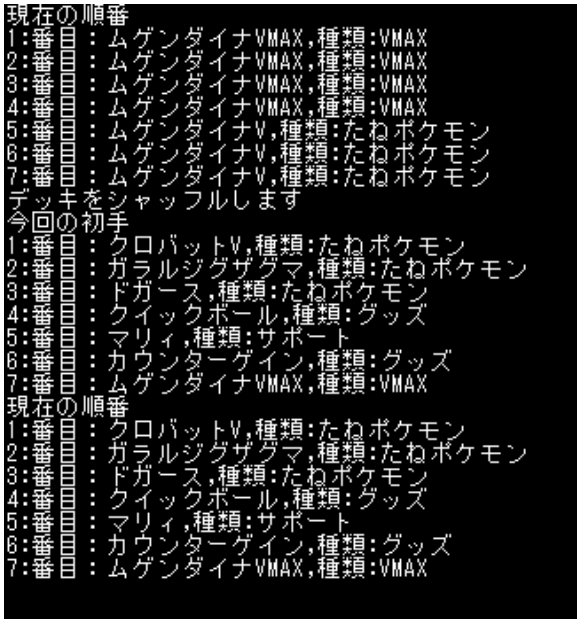

図 7firstOfSeven メソッド実行例

3.2.3. putSide メソッド

putSide メソッドは,デッキからサイドを置くメソッドである.初手を引いた後にデッキをシャッフルしてはいけないため,このメソッドではシャッフルは行わない.図 8 は putside メソッドの実行とその前後でのデッキの順番をを示す.図 8 より,putSide メソッドではデッキのシャッフルを行っていないことがわかる.

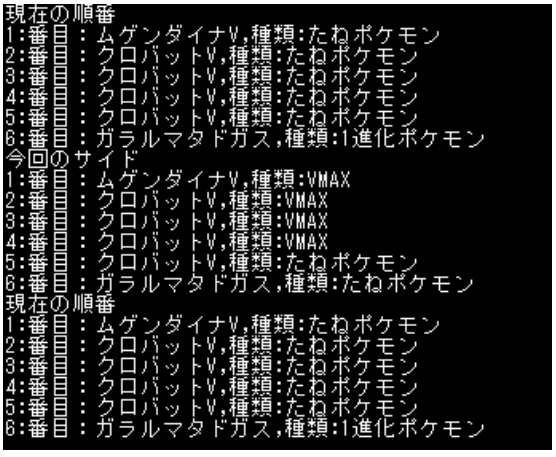

図 8putSide メソッド実行例

3.2.4. wantOne メソッド

wantOne メソッドは,入力されたカードが初手の 7 枚にどれだけの頻度で含まれているかを算出するメソッドである.このメソッドでは,たねポケモンがない場合を考慮していない.図 9 では,wantOne メソッドを実行し各回での手札を表示したのち入力したカードを引いた回数を表示している.

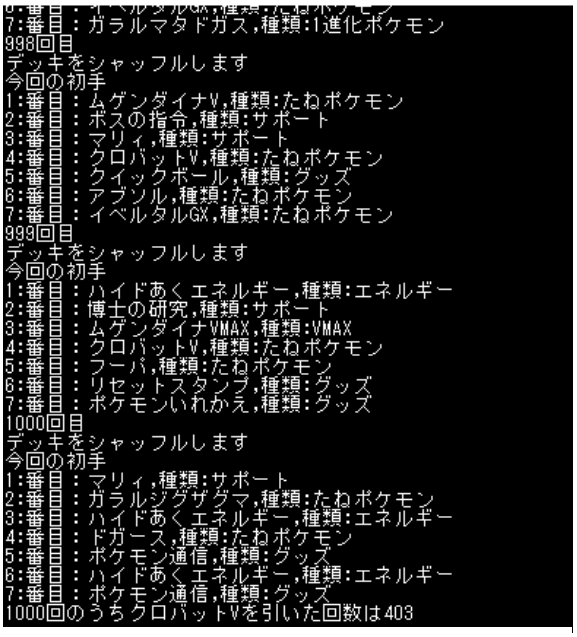

図 9wantOne 実行例

3.2.5. wantOneWithoutOther メソッド

wantOneWithoutOther メソッドは,入力されたカードが初手の 7 枚にどれだけの頻度で含まれているかをたねポケモンがなかった場合を含めず算出するメソッドであり,wantOne メソッドが元になってる.図 10 では,wantOneWithoutOther メソッドを実行し各手札を表示し,入力したカードとマリガンした回数,実際の回数を表示している.

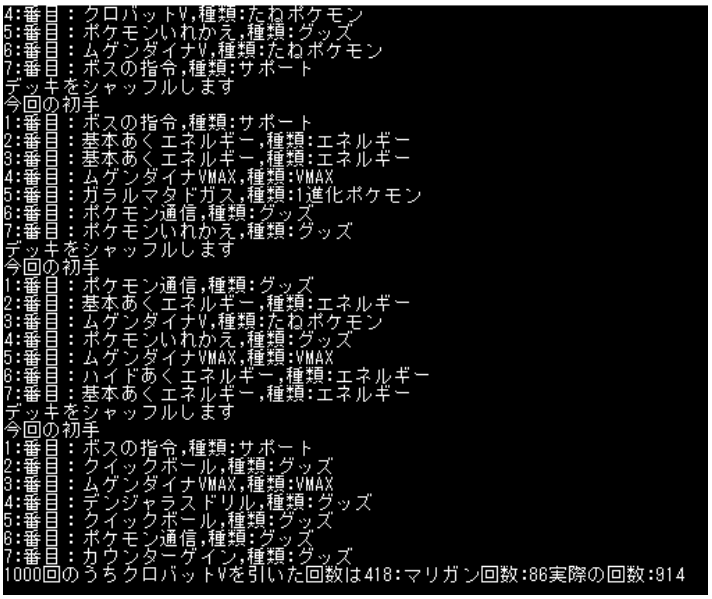

図 10wantOneWithoutOther 実行例

3.2.6. intoSide メソッド

intoSide メソッドは,入力されたカードが 6 枚のサイドの中にどれだけの頻度で含まれているか算出するメソッドである.最初の手札にたねポケモンがなかった場合を考慮していない.図 11 では,intoSide メソッドを実行し,各回のサイドのカードと入力したカードがサイドに含まれた回数を表示している.

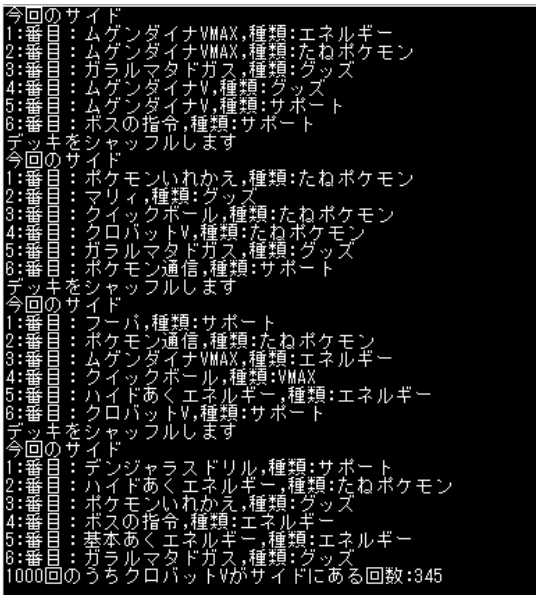

図 11intoSide 実行例

3.2.7. intoSideWithoutOther メソッド

intoSideWithoutOther 入力されたカードが 6 枚のサイドの中にどれだけの頻度で含まれているかたねポケモンがいない場合を考慮して算出するメソッドであり,intoSide メソッドを元にしている.図 12 では,intoSideWithoutOther メソッドを実行し,各回のサイドのカードと入力したカードがサイドに含まれた回数,マリガン回数を表示している.

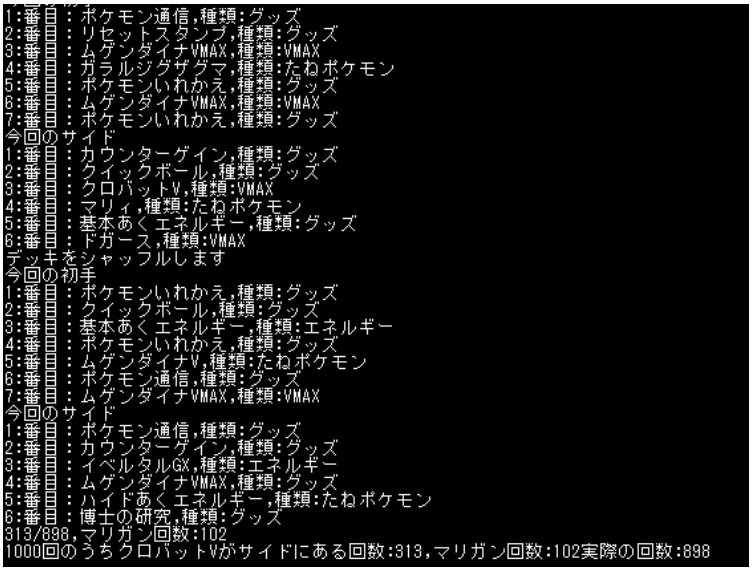

図 12intoSideWithoutOther 実行例

3.2.8. dontStartOne メソッド

dontStartOne メソッドは,入力されたカードが最初の手札に含まれた回数と入力したカードでゲームを始めた回数を算出するメソッドであり,wantOne メソッドを元にしている.図 13 では,dontStartOne メソッドを実行し,各回での手札のカードと入力したカードを引いた回数,そのカードでゲームを始めた回数を表示している.

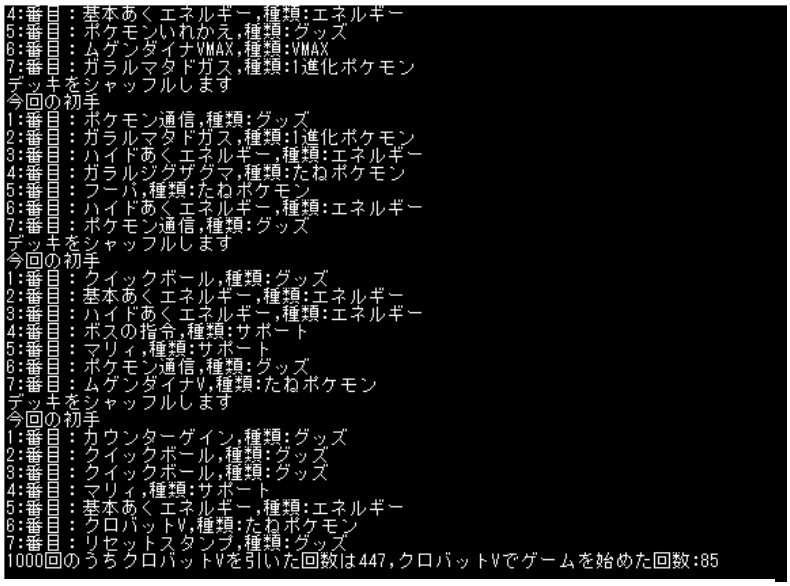

図 13dontStartOne 実行例

3.2.9. wantOneWithoutOtherAndDraw メソッド

wantOneWithoutOtherAndDraw メソッドは,入力されたカードが最初の手札とゲームが始まった最初のドローに含まれた回数をたねポケモンがない場合を考慮して算出するメソッドである.wantOne メソッドを元にしている.図 14 は wantOneWithoutOtherAndDraw メソッドを実行し,各回の初手手札と最初のドロー,マリガン回数を表示している.

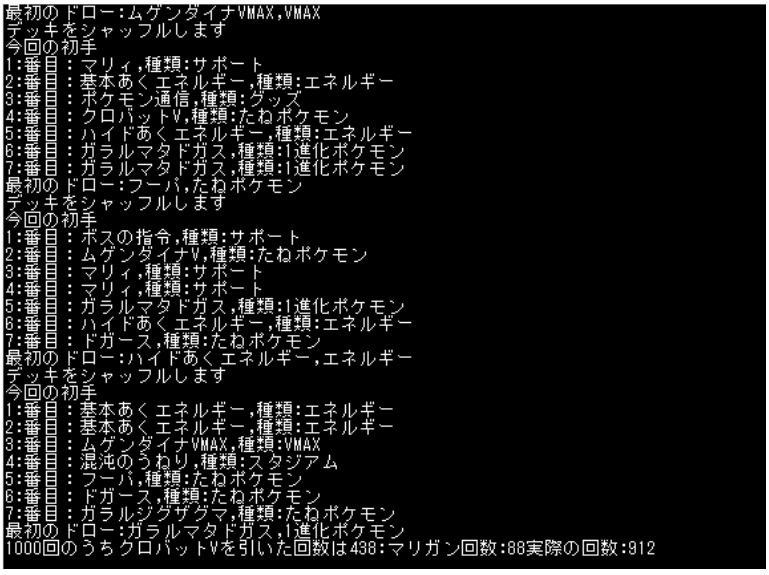

図 14wantOneWithoutOtherAndDraw 実行例

3.2.10. main メソッド

main メソッドは,このプログラムを実行する時に最初に呼び出され,3.1 章で記載した出力を行う.

3.3. SixtyDeckMaker.java

SixtDeckMaker クラスは,ユーザから入力されたデータを元にデッキを作るクラスである.

3.4. EternatusVMAX.java

EternalVMAX クラスあらかじめデータが入力されデッキが実装されるクラスである.

4. 検証内容と考察

この章では,本研究で作成したプログラムの検証を行う.wantOne メソッドと intoSide メソッドを充分な回数試行し算出した結果と理論値を比較し,精度の検証を行う.

4.1. 理論値の算出

wantOne メソッドと intoSide メソッドの理論値は以下の式から求められる.

$$\frac{{}_{60}C_n - {}_{60-m}C_n}{{}_{60}C_n}$$

n:場所によるカードの枚数(初手の手札なら 7,サイドなら 6)
m:入力したカードの枚数

4.2. 比較と考察

前節の数式で求めた理論値とプログラムによる出力結果(入力したカードの枚数は 4 枚,試行回数 1000000 回)を以下に示す.充分な試行回数の出力結果と理論値が大きくずれていないことから,本研究で作成したプログラムは精度が高いと判断できる.

また,既存の補助サービスでは算出されない少ない回数の際に起こりうるデータの偏りもこのプログラムでは算出することができる.これらのことから,人力や既存のサービスの代わりにこのプログラムでデータを集めても問題ないと考えられる.

表 2 出力結果と理論値の比較

	プログラムによる算出	理論値
入力したカードが初手に含まれる確率	39.94%	39.94%
入力したカードがサイドに含まれる確率	35.15%	35.14%

5. 結論と今後の課題

本研究では,Java を用いてポケモンカードのデッキ作成を補助するプログラムを作成し,充分な出力結果を確認でき,また個人で使う分には問題なく動作した.今後の課題としては,より複雑な条件での確率の算出や,他者に使用してもらい使用感を調査し,より使いやすくするためのユーザインタフェースの改善,他のカードゲームルールでの実装等があげられる.

6. 謝辞

卒業研究のテーマ決めやレジュメや卒業論文の推敲等, 石水隆講師には多方面で指導賜りました. ここに感謝の意を表します.

文献目録

[1] Cygames, Shadowverse, https://shadowverse.jp/.

[2] Blizzard Entertainment, *Hearthstone,* 2014.

[3] HSReplay.net, *HearthstoneDeckTracker.*

[4] 株式会社ブシロード, “DECKLOG,” [オンライン]. Available: https://decklog.bushiroad.com/. [アクセス日: 30 1 2021].

[5] (株)クリーチャーズ,(株)ポケモン, “デッキ構築 | ポケモンカードゲーム,” [オンライン]. Available: https://www.pokemon-card.com/deck/. [アクセス日: 30 1 2021].

[6] 深津貴之, “MTG 等,カードゲーム汎用の確率計算シート,” 28 8 2020. [オンライン]. Available: https://note.com/fladdict/n/n7939e60fdf2f. [アクセス日: 30 1 2021].

[7] (株)クリーチャーズ,(株)ポケモン, “はじめよう!ポケモンカード！,” [オンライン]. Available: https://www.pokemon-card.com/about/. [アクセス日: 29 1 2021].

[8] 株式会社ポケモン, “数字で見るポケモン,” [オンライン]. Available: https://corporate.pokemon.co.jp/aboutus/figures/. [アクセス日: 29 1 2021].

[9] “MAGIC THE GATHERING 日本公式ウェブサイト,” Wizards of the Coast LLC, [オンライン]. Available: https://mtg-jp.com/. [アクセス日: 29 1 2021].

[10] Wizards of the Coast LLC, “マジック 20 年の歩み,” [オンライン]. Available: https://mtg-jp.com/20th/history.html. [アクセス日: 30 1 2021].

[11] Wizards of the Coast LLC, “デッキの作り方(MAGIC THE GATHERING),” [オンライン]. Available: https://mtg-jp.com/howtoplay/phase7/. [アクセス日: 30 1 2021].

[12] (株)クリーチャーズ,(株)ポケモン, “はじめてのデッキ作り,” [オンライン]. Available: https://www.pokemon-card.com/deck/first-deck/. [アクセス日: 30 1 2021].

[13] 株式会社コナミデジタルエンタテインメント, “マスタールール対応公式ルールブック,” [オンライン]. Available: https://img.yugioh-card.com/japan/howto/data/rulebook_masterrule20200401_ver1.0.pdf. [アクセス日: 30 1 2021].

[14] 株式会社タカラトミー, “読んでルールをおぼえよう! | デュエル・マスターズ,” [オンライン]. Available: https://dm.takaratomy.co.jp/rule/basic/basic08/. [アクセス日: 30 1 2021].

[15] 株式会社コナミデジタルエンタテインメント, “遊戯王 OCG デュエルモンスターズ,” [オンライン]. Available: https://www.yugioh-card.com/japan/. [アクセス日: 30 1 2021].

[16] コナミホールディングス株式会社, *遊☆戯☆王アーク・ファイブ TAGFORCESPECIAL,* 2015.

[17] Cygames Research 佐藤 , “ゲーム AI 実践編-Shadowverse に見る TCGAI 開発の事例 1,” 27 7 2016. [オンライン]. Available: https://tech.cygames.co.jp/archives/2853/. [アクセス日: 30 1 2021].

[18] (株)クリーチャーズ,(株)ポケモン, “ポケモンカードゲームのあそびかた,” [オンライン]. Available: https://www.pokemon-card.com/rules/howtoplay/. [アクセス日: 30 1 2021].

[19] (株)クリーチャーズ,(株)ポケモン, “カードの種類と見かた,” [オンライン]. Available: https://www.pokemon-card.com/rules/howtoplay/basic_rules/01.html. [アクセス日: 30 1 2021].

[20] (株)クリーチャーズ,(株)ポケモン, “カード検索 | ポケモンカードゲーム公式ホームページ,”

[オンライン]. Available: https://www.pokemon-card.com/card-search/. [アクセス日: 1 2 2021].

- 付録 ソースコード

今回作成したプログラムのソースコードを下記に示す.

1. DeckAssist.java

```
import java.util.Random;
import java.util.Scanner;
import java.util.concurrent.ThreadLocalRandom;
import java.util.Arrays;

public class DeckAssist {
	public String[][]deck;//要素1はカード名,要素2はカードの種類
	public String[][]firstHand;//初手手札を格納
	public String[][]anotherZone;//サイドを格納
	public double counter=0.00;//算出した値を格納

	public    void shuffle() {//デッキをシャッフルするメソッド
	  // 配列が空か 1 要素ならシャッフルしようがないので、そのままreturn
		if(deck.length<=1) {
			return;
		}
		System.out.println("デッキをシャッフルします");
		Random rnd = ThreadLocalRandom.current();
		for(int i=deck.length-1;i>0;i--) {
			int index = rnd.nextInt(i+1);
			//要素の入れ替え
			String tmp = deck[index][0];
			String tmp2 = deck[index][1];//要素が2つあるので2つ分宣言
			deck[index][0]=deck[i][0];
			deck[index][1]=deck[i][1];
			deck[i][0]=tmp;
			deck[i][1]=tmp2;
		}
	}
```

```
public void firstOfSeven() {//初手手札を格納する
		shuffle();
		firstHand = new String[7][2];
		for(int n=0;n<7;n++) {
			firstHand[n][0]=deck[n][0];
			firstHand[n][1]=deck[n][1];
		}
		System.out.println("今回の初手");
		for(int n=0;n<7;n++) {
			System.out.println(n+1+":番目 : "+deck[n][0]+",種類:"+deck[n][1]);
		}
		//System.out.println(Arrays.deepToString(firstHand));
		//確認したときにコメントを外す
}

public void putSide() {//サイドを格納する,サイドは初手引いてから置くためシャッフルはしない
		anotherZone = new String[6][2];
		for(int i=0;i<6;i++) {
			anotherZone[i][0]=deck[7+i][0];
			anotherZone[i][1]=deck[7+i][1];
		}
		System.out.println("今回のサイド");
		for(int n=0;n<6;n++) {
			System.out.println(n+1+":番目 : "+anotherZone[n][0]+",種類:"+deck[n][1]);
		}
		//System.out.println(Arrays.deepToString(anotherZone));
		//確認したいときにコメントを外す
}

public void wantOne() {//初手に来てほしいカードが来る回数
		System.out.println("カード名を入力");
		Scanner scan = new Scanner(System.in);
		String str = scan.next();

		int count = 0;
		for(int i=0;i<1000;i++) {
			System.out.println(i+1+"回目");
			firstOfSeven();
			for(int hand=0;hand<7;hand++) {
```

```
                        if(firstHand[hand][0].equals(str)) {
                                count++;
                                break;
                        }else {
                        }
                }
        }
        System.out.println("1000回のうち"+str+"を引いた回数は"+count);
        counter = count;
        //値をcounterに格納しゲッターで表示できるようにする
}


public void wantOneWithoutOther() {//初手に来てほしいカードが来る回数,マリガンは数えない
        System.out.println("カード名を入力");
        Scanner scan = new Scanner(System.in);
        String str = scan.next();

        int count = 0; //欲しいカードが引けた回数
        int countP = 0;//たねポケモンの枚数
        int realCount = 0;//試行回数の総数
        int beforeCount = 0;
        for(int i=0;i<1000;i++) {
                firstOfSeven();
                for(int hand=0;hand<7;hand++) {//欲しいカードを引けているか確認
                        if(firstHand[hand][0].equals(str)) {
                                count++;
                                break;
                        }else {
                        }

                }
                realCount++;
                for(int hand=0;hand<7;hand++) {
                        if(firstHand[hand][1].equals("たねポケモン")) {
                                countP++;
                                break;
                        }else {
                        }
```

```
                }
                if(countP==0) {//マリガン回数分総数から減らす
                        realCount--;
                }
                if(countP==0&&beforeCount<count) {//ポケモンがなく、欲しいカードも引けていない場合余計に一回countが小さくなるのを防ぐ
                        count--;
                }
                countP=0;
                beforeCount = count;
        }
        System.out.println("1000回のうち"+str+"を引いた回数は"+count+":マリガン回数:"+(1000-realCount)+"実際の回数:"+realCount);
    }

    public void intoSide() {//サイドに落ちる回数　を求める
        System.out.println("カード名を入力");
        Scanner scan = new Scanner(System.in);
        String str = scan.next();

        int count = 0;
        for(int i=0;i<1000;i++) {
                shuffle();
                putSide();
                for(int hand=0;hand<6;hand++) {
                        if(anotherZone[hand][0].equals(str)) {
                                count++;
                                break;
                        }else {
                        }
                }
        }

        counter = count;
        System.out.println("1000回のうち"+str+"がサイドにある回数:"+count);
    }

    public void intoSideWithoutOther() {//サイドに落ちる回数,マリガンは含めない
        System.out.println("カード名を入力");
```

```
Scanner scan = new Scanner(System.in);
String str = scan.next();

int count = 0; //欲しいカードが引けた回数
int countP = 0;//たねポケモンの枚数
int realCount = 0;//試行回数の総数
int beforeCount = 0;//直前のcount
int dontStart = 0;
for(int i=0;i<1000;i++) {
		firstOfSeven();
		putSide();
		for(int side=0;side<6;side++) {//欲しいカードを引けているか確認
			if(anotherZone[side][0].equals(str)) {
				count++;
				break;
			}else {
			}

		}
		realCount++;
		for(int hand=0;hand<7;hand++) {
			if(firstHand[hand][1].equals("たねポケモン")) {
				countP++;
				break;
			}else {
			}
		}
		if(countP==0) {//マリガン(たねポケモンがない)回数分総数から減らす
			realCount--;
			dontStart++;
		}
		if(countP==0&&beforeCount<count) {//ポケモンがなく、欲しいカードも引けていない場合余計に一回countが小さくなるのを防ぐ
			count--;
		}
		countP=0;
		beforeCount = count;
}
```

```
                System.out.println("1000回のうち"+str+"がサイドにある回数:"+count+",マリガン回数:"+dontStart+"実際の回数:"+(1000-dontStart));
        }

        public void dontStartOne() {//入力したカードが初手手札に来てそれ以外でスタートできるか
                System.out.println("カード名を入力");
                Scanner scan = new Scanner(System.in);
                String str = scan.next();
                int count = 0;
                int beforeCount = 0;//前回のカウント数
                int dStartOne = 0;//スタートしたくないポケモン
                int bPokemon = 0;//たねポケモンの数
                int startOne=0;//スタートしたくないポケモンで始まった回数
                for(int i=0;i<1000;i++) {
                        beforeCount=count;
                        firstOfSeven();
                        for(int hand=0;hand<7;hand++) {
                                if(firstHand[hand][0].equals(str)) {
                                        count++;
                                }else {
                                }
                        }
                        if(count>beforeCount) {//入力したカードがなければそこで終わり
                        for(int hand=0;hand<7;hand++) {
                                if(firstHand[hand][0].equals(str)) {
                                        dStartOne++;
                                }else {
                                }

                        if(firstHand[hand][1].equals("たねポケモン")) {
                                bPokemon++;
                        }else {
                        }
                        }
                        if(dStartOne==bPokemon) {
                                startOne++;
                        }
                        }
                        dStartOne=0;//それぞれ値を0に戻す
```

```
                    bPokemon=0;//

                }
                System.out.println("1000回のうち"+str+"を引いた回数は"+count+","+str+"でゲームを始めた回数:"+startOne);
        }

        public void wantOneWithoutOtherAndDraw() {//入力したカードが初手手札+最初のドローで来る回数
                System.out.println("カード名を入力");
                Scanner scan = new Scanner(System.in);
                String str = scan.next();

                int count = 0; //欲しいカードが引けた回数
                int countP = 0;//たねポケモンの枚数
                int realCount = 0;//試行回数の総数
                int beforeCount = 0;//直前のcount
                for(int i=0;i<1000;i++) {
                        firstOfSeven();
                        System.out.println("最初のドロー:"+deck[13][0]+","+deck[13][1]);
                        for(int hand=0;hand<8;hand++) {//欲しいカードを引けているか確認
                                if(hand<7) {
                                if(firstHand[hand][0].equals(str)) {
                                        count++;
                                        break;
                                }else {
                                }
                                }else {
                                        if(deck[hand+6][0].equals(str)) {
                                                count++;
                                                break;
                                        }else {
                                        }
                                }
                        }
                        realCount++;
                        for(int hand=0;hand<7;hand++) {
                                if(firstHand[hand][1].equals("たねポケモン")) {
                                        countP++;
                                        break;
```

```
                    }else {
                    }
                }
                if(countP==0) {//マリガン(たねポケモンがない)回数分総数から減らす
                    realCount--;
                }
                if(countP==0&&beforeCount<count) {//ポケモンがなく、欲しいカードも引けていない場合余計に一回countが小さくなるのを防ぐ
                    count--;
                }
                countP=0;
                beforeCount = count;
            }
            counter = count;
            System.out.println("1000回のうち"+str+"を引いた回数は"+count+":マリガン回数:"+(1000-realCount)+"実際の回数:"+realCount);

        }


        public void nowSort() {
            System.out.println("現在の順番");
            for(int n=0;n<6;n++) {
                System.out.println(n+1+":番目："+deck[n+7][0]+",種類:"+deck[n+7][1]);
            }
        }
        public double getCounter() {
            return counter;
        }
        public static void main(String[] args) {
            //deck = new SixtyDeckmaker();
            //入力して使いたい場合はコメントを外す
            EternatusVMAX deck = new EternatusVMAX();
            System.out.println("0から6のいずれかの数字を入力してください");
            System.out.println("0:初手を調べる");
            System.out.println("1:マリガンを除いて初手を調べる");
            System.out.println("2:サイドを調べる");
            System.out.println("3:マリガンを除いてサイドを調べる");
            System.out.println("4:入力したカードでゲームを開始しないかを調べる");
```

```
                System.out.println("5:初手と最初のドローを調べる");
                System.out.println("6:現在のデッキの並び順を調べる");

                Scanner scan = new Scanner(System.in);
                int str = scan.nextInt();

                switch(str) {
                case 0:
                        deck.wantOne();
                        break;
                case 1:
                        deck.wantOneWithoutOther();
                        break;
                case 2:
                        deck.intoSide();
                        break;
                case 3:
                        deck.intoSideWithoutOther();
                        break;
                case 4:
                        deck.dontStartOne();
                        break;
                case 5:
                        deck.wantOneWithoutOtherAndDraw();
                        break;
                case 6:
                        deck.nowSort();
                        break;
                default:
                System.out.println("0から6の数字を入力してください");
                break;
                }

        }

}
```

2. SixtyDeckmaker.java

```
import java.util.Scanner;
```

```
public class SixtyDeckmaker extends DeckAssist{

        public void makeadeck() {
                deck = new String[60][2];//要素1はカード名,要素2はカードの種類
                for(int i=0;i<60;i++) {

                        System.out.println("カード名を入力");
                        Scanner scan = new Scanner(System.in);
                        String str = scan.next();

                        System.out.println("種類名を入力");
                        Scanner scan3 = new Scanner(System.in);
                        String syurui = scan3.next();

                        System.out.println("枚数を入力");
                        Scanner scan2 = new Scanner(System.in);
                        int num = scan2.nextInt();
                        if(num+i<=60) {
                                while(num>0) {
                                        deck[i][0]=str;
                                        deck[i][1]=syurui;
                                i++;
                                        num--;
                                }
                                i--;
                                }else{
                                        System.out.println("超過分は入りません");
                                        while(i<60) {//デッキの残りを同じカード追加するwhile文
                                                deck[i][0]=str;
                                                deck[i][1]=syurui;
                                                i++;
                                        }
                                }

                }
                int count = 0;
                while(count<60) {
                        count++;
```

```
        }

    }

}
```

3. EternatusVMAX.java

```
public class EternatusVMAX extends DeckAssist {
    public EternatusVMAX() {
        deck = new String[60][2];
        deck[0][0]="ムゲンダイナVMAX";
        deck[1][0]="ムゲンダイナVMAX";
        deck[2][0]="ムゲンダイナVMAX";
        deck[3][0]="ムゲンダイナVMAX";
        deck[4][0]="ムゲンダイナV";
        deck[5][0]="ムゲンダイナV";
        deck[6][0]="ムゲンダイナV";
        deck[7][0]="ムゲンダイナV";
        deck[8][0]="クロバットV";
        deck[9][0]="クロバットV";
        deck[10][0]="クロバットV";
        deck[11][0]="クロバットV";
        deck[12][0]="ガラルマタドガス";
        deck[13][0]="ガラルマタドガス";
        deck[14][0]="ガラルマタドガス";
        deck[15][0]="ドガース";
        deck[16][0]="ドガース";
        deck[17][0]="ドガース";
        deck[18][0]="アブソル";
        deck[19][0]="フーパ";
        deck[20][0]="ガラルジグザグマ";
        deck[21][0]="ガラルジグザグマ";
        deck[22][0]="イベルタルGX";
        deck[23][0]="クイックボール";
        deck[24][0]="クイックボール";
        deck[25][0]="クイックボール";
        deck[26][0]="クイックボール";
        deck[27][0]="ポケモン通信";
        deck[28][0]="ポケモン通信";
```

```
        deck[29][0]="ポケモン通信";
        deck[30][0]="ポケモン通信";
        deck[31][0]="ポケモンいれかえ";
        deck[32][0]="ポケモンいれかえ";
        deck[33][0]="ポケモンいれかえ";
        deck[34][0]="ポケモンいれかえ";
        deck[35][0]="リセットスタンプ";
        deck[36][0]="リセットスタンプ";
        deck[37][0]="カウンターゲイン";
        deck[38][0]="カウンターゲイン";
        deck[39][0]="デンジャラスドリル";
        deck[40][0]="博士の研究";
        deck[41][0]="博士の研究";
        deck[42][0]="博士の研究";
        deck[43][0]="マリィ";
        deck[44][0]="マリィ";
        deck[45][0]="マリィ";
        deck[46][0]="ボスの指令";
        deck[47][0]="ボスの指令";
        deck[48][0]="ボスの指令";
        deck[49][0]="混沌のうねり";
deck[50][0]="基本あくエネルギー";
        deck[51][0]="基本あくエネルギー";
        deck[52][0]="基本あくエネルギー";
        deck[53][0]="基本あくエネルギー";
        deck[54][0]="基本あくエネルギー";
        deck[55][0]="基本あくエネルギー";
        deck[56][0]="ハイドあくエネルギー";
        deck[57][0]="ハイドあくエネルギー";
        deck[58][0]="ハイドあくエネルギー";
        deck[59][0]="ハイドあくエネルギー";
        int i = 0;
        while(i<4) {
                deck[i][1]="VMAX";
                i++;
        }
        while(i<12) {
                deck[i][1]="たねポケモン";
                i++;
```

```
                }
                while(i<15) {
                        deck[i][1]="1進化ポケモン";
                i++;
                }
                while(i<23) {
                        deck[i][1]="たねポケモン";
                i++;
                }
                while(i<40) {
                        deck[i][1]="グッズ";
                i++;
                }
                while(i<49) {
                        deck[i][1]="サポート";
                        i++;
                }
                        while(i<50) {
                        deck[i][1]="スタジアム";
                        i++;
                        }
                while(i<60) {
                        deck[i][1]="エネルギー";
                i++;
                }
        }
}
```