卒業研究報告書

題目

ジャンケン将棋アプリの開発

指導教員

石水 隆　講師

報告者

09-1-037-0183

南野 裕介

近畿大学理工学部情報学科

平成 27 年 1 月 31 日提出

概要

本研究はジャンケン将棋[9]のプログラムの作成を目的としたものである。ジャンケン将棋は二人零和有限確定完全情報ゲームであり、立方体に描かれたジャンケンの優劣で駒もしくはゴールを取り合う二人用のボードゲームである。本来ジャンケン将棋は 6×6 マスのゲーム盤で双方４個ずつ駒を使い行うが、本研究では駒を双方 1 個ずつに制限する。また、本来では各手番で 2 手動かせるが、本研究では簡単のために各手番では 1 手のみ動かせるものとした簡易版ジャンケン将棋のプログラムの作成とする。

目次

1　序論

1.1 二人零和有限確定完全情報ゲーム

囲碁や将棋、チェスなどといった代表されるボードゲームは二人零話有限確定完全情報ゲームであり、本研究のジャンケン将棋[9]も二人零話有限確定完全情報ゲームに分類される。二人とはゲームを行う人数が二人、または二つのグループでする事である。零和とは全プレイヤーの利得合計が０となる。例えば、勝利得点１・敗北得点－１・引き分け得点０としたら、いつでも全プレイヤーの合計得点が０となる事である。有限とは各プレイヤーの手の組合せが有限であるゲームである。確定とは運などの偶然の要素がない事である。完全情報とは各プレイヤーが相手の行ったプレイ内容を自身が知る事ができる事である。

1.2 二人零和有限確定完全情報ゲームの完全解析に関する既知の結果

二人零和有限確定完全情報ゲームは、理論上は完全先読みが可能であり双方のプレイヤーが最善手を打てば、必ず先手必勝か後手必勝か引き分けかのいずれかに決まる。しかし、完全先読みをするには選択する局面が膨大であればあるほど困難である。この事から多くのボードゲームは完全解析が不可能なケースが多い。たとえば、総局面数がそれぞれリバーシが10^{28}通り・チェスが10^{50}通り・将棋が10^{69}通り・囲碁が10^{170}通りずつあるとされている。これらの総局面数は現在の計算機では計算不可能である。

一方、既に解析されているボードゲームも存在している。たとえば、連珠(五目並べの種類)は双方最善手を打った場合、47 手で先手必勝である[1]。チェッカーは双方最善手を指すと引き分けとなる[2]。また局面数の多い将棋や囲碁はゲーム盤の大きさを小さくすることで解析されている。例として、将棋はゲーム盤のサイズ3×4 に駒の種類を4 種類に減らした「どうぶつしょうぎ」[3]が完全解析されており、双方最善手を指すと78 手で後手必勝が判明している[4]。同様に、ゲーム盤サイズを×5、駒を3種類、持ち駒無しに制限した「パンマンはじめて将棋」[7]は双方最善手を指すと千日手で引き分けとなる[8]。図1，2に「どうぶつしょうぎ」およびアンパンマンはじめて将棋の盤と駒の初期配置を示す。囲碁はゲーム盤のサイズ4×4 に減らした場合、双方最善手を打つと引き分けに[5]、サイズ5×5 に減らした場合、双方最善手を打つと25 目で先手必勝となる[6]。

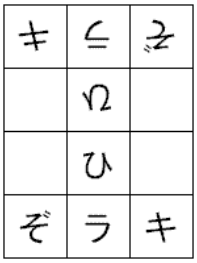


図 1 どうぶつしょうぎの盤と駒の初期配置

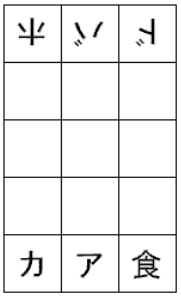


図 2 アンパンマンはじめて将棋の盤と駒の初期配置

1.3 ゲーム AI の手法

可能な局面数が多いゲームに対して完全解析を行うことは困難である。そのようなゲームに対しては確実な最適手を得ることはできないが、局面の評価値計算、定跡データベース、一定手数の先読み、終盤での必勝読みと完全読み、モンテカルロ法などを用いてより有利だと思われる手を選択することができる。

局面の評価計算の手法は、局面が自分にとって有利か不利かを判断し、局面の評価を行い、数字として表わすことで選択肢の中から自分にとって有利なものを選ぶ手法である。AIの強さは評

価関数の作り方に応じて決まるため、評価関数はできるだけ戦局を適切に評価できるように工夫して作る必要がある。

定跡データベースを用いる手法は、定跡データベースの中から自分の同じ局面のもの参照し、その結果から自分の次の着手を決める手法である。定跡データベースを使用することで、強い手を選択することができる。しかし、相手があえて定跡以外の手を指すなどして、データベースに無い局面が出てきたときにはこの手法は使えない。

一定手数の先読みは、手数を制限せずの先読みは膨大になり不可能になるが、一定に決めることにより先読みが可能になり、その結果から決める。一般に先読みする手数が多いほど強いAIとなるが、先読み手数の増加に伴い探索時間が指数的に増えるため、適度に枝刈りをして探索範囲を減らす工夫をする必要がある。

終盤での必勝読みと完全読みは、局面が終盤になるにつれて各プレイヤーの着手は少なくなるので、先読みが深く行える。先読みの結果により、必勝の一手を決める。終盤での読みは、必勝読みと完全読みがある。必勝読みとはゲーム終盤で勝敗のみを読み切り、必ず勝てる手を指すことを言う。完全読みとは、そこから得られる全ての局面を読み、最も点数の高くなる手を指すことを言う。必勝読みの方が計算時間が少なくてすむため、一般にまず必勝読みで勝ちを確定させた上で、残り手数が少なくなると完全読みに切り替えてより点数の高い勝ちを目指すことが多い。

モンテカルロ法は、互いの手をランダムに行うゲームを何回もする。この何回も行われたゲームの結果により評価値を決定し、その評価値で自分の着手を決める。

以上の手法を用いることにより、完全解析を行わなくてもある程度の強さのプログラムを作ることが可能であり、ゲームによってはプロに勝つこともできる。

1.4 ジャンケン将棋

ジャンケン将棋[9]は、対戦人数は二人で行い、グー・チョキ・パーが描かれたサイコロ形の駒を転がして移動し、駒に描かれたじゃんけんで勝てば相手のコマを取ることができる。相手駒をすべて取るか、相手のゴールに自駒を入れば勝つボードゲームである。

1.5 本報告書の構成

本報告では、まず2章で本研究の対象であるジャンケン将棋について説明する。続く第3章で、本研究で作成したジャンケン将棋プログラムについて述べる。4章において結果を示し、また考察を行う。最後に5章で結論および今後の課題を述べる。

2 ジャンケン将棋

本章では、ジャンケン将棋[9]について説明する。

2.1 ジャンケン将棋の概要

ジャンケン将棋は熊本県立八代工業高等学校インテリア科教師の梅田龍一氏によって考案され、学研の「頭のよくなるゲームシリーズ」として 2010 年に発売された。ジャンケン将棋は立方体に描かれたじゃんけんの優劣で駒もしくはゴールを取り合う 2 人用ボードゲーム(図 3)である。使用するものは、６×６マスのゲーム盤(図 4)、６つの面にグー・チョキ・パーの三種類が二つずつ描かれた正方形の駒を８つ(図 5)、１～３までの描かれたサイコロの以上の三点である。

図 3　ジャンケン将棋本体[9]

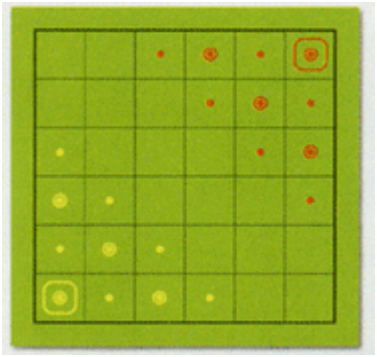


図 4　ゲーム盤[9]

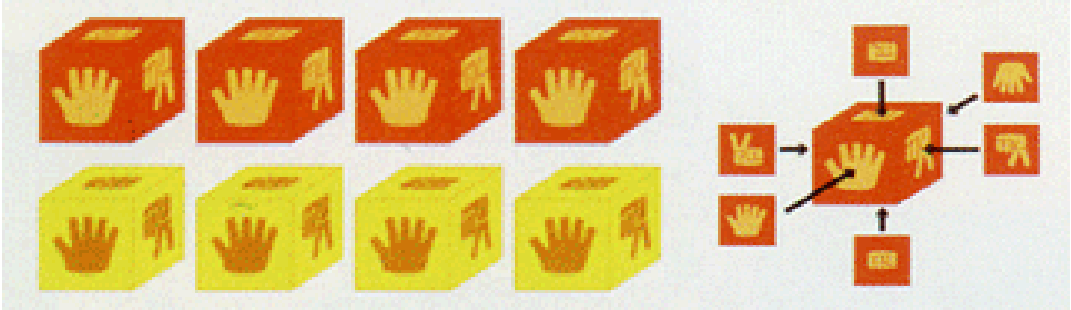


図 5　ジャンケン将棋の駒[9]

2.2 ジャンケン将棋のルール

ジャンケン将棋のルールは以下の通りである。

勝利条件：相手駒を全滅させるか自駒を相手のゴールにピッタリ入れば勝ちになる。ゴールの位置は図 2 のゲーム盤のオレンジの点を四角く囲んでいる角の所と対角線上にある黄色の点を四角く囲んでいる角の所の２か所にある。

準備：駒の初期配置位置は決まっており、図 4 のゲーム盤の大きめの点のところに配置する。なお、配置時の駒の向きはグー、チョキ、パーどの面を上面にしてもいい。

駒の動かし方：駒は基本回転して移動する。駒を動かせる数の事を行動力という。各手番では初めから行動力が２ある。１つの駒を２行動力を使用するか２つの駒を１行動力ずつ使用するのどちら選択するができる。また同じ駒を動かして元の場所に戻る事はできない。

駒の取り方：自駒の上下左右の１マスに相手駒があり、また自駒の上面のジャンケンの手がその相手駒の上面のジャンケンの手に勝てば、行動力１使い回転せずに取ることができる。

出戻り禁止エリア：自分のゴールから３マス離れた所までが自分のエリアになる。図２でいうと自分がオレンジならオレンジの点の所すべて自分のエリアになる。一度、自駒がここから出てしまうともうその駒は自分のエリアには戻れない。
ふんばりモード：自駒が最後の１つになると自分の手番の行動力が２から３になる。また本来なら出戻り禁止エリアから出るともう戻れないがこのモードの時は自由になる。

2.3 簡易版ジャンケン将棋

簡易版ジャンケン将棋は本来のジャンケン将棋のルールや駒の数を変更したものである。本来ならジャンケン将棋の双方のプレイヤーの持ち駒は４個ずつで簡易版では１個ずつに減らし、行動力も２から１にした。本来なら初期配置の際、駒の上面は選べるが簡易版は上面が決まっている。勝利条件も本来なら相手のゴールに自駒を配置できると勝利というのがあったのが、簡易版はコレを無くし相手駒を取ると勝利だけにした。本来なら駒が移動すると必ず回転するが、簡易版は駒を回転させてか回転せずそのまま移動するか選択できるようにした。

また本研究の目的がジャンケン将棋のプログラムの作成であるため、その初期段階として簡易版ジャンケン将棋のプログラムの作成をした。

3 研究内容

本章では本研究で作成した簡易版ジャンケン将棋のプログラムについて説明する。

本研究では、Java を用いてジャンケン将棋プログラムを作成した。付録に本研究で作成したプログラムのソースを示す。

本研究で作成したジャンケン将棋プログラムは、JanShogi クラス、Board クラス、Constants インタフェースの 3 つから成る。

3.1 JanShogi クラス

JanShogi クラスはジャンケン将棋プログラムの中心となるクラスであり、ジャンケン将棋の基本動作のメソッド二つとメインメソッドで構成されている。以下に各メソッドについて述べる。

- **void player()**
  player()は手番プレイヤーが一手動かし、勝利判定するメソッドである。プレイヤーに自駒をどちらの方向に移動させるかを選択させ、もし移動不可能な位置ならばエラーを出し再度入力を促す。移動可能なら駒を移動するが、この時に駒を回転させるかそのまま回転せず移動するかの選択ができる。移動先に相手駒がある場合、上面の手により勝敗判定が行われ、勝負が付けば勝者を表示して終了、引き分けの場合は続行となる。
- **void selectPiece()**
  selectPiece()は駒を選択するメソッドである。プレイヤーは盤上の座標を入力する。その座標に自駒があればその駒を選択し、その座標が空きマス、または相手駒がある場合は再度入力を促す。
- **メインメソッド**
  プログラムを実行するのに不可欠なメソッドであり、実行された際は最初に呼び出され、メインメソッドの命令内容が実行される。

3.2 Board クラス

Board クラスはジャンケン将棋の盤面を管理するクラスである。

- **void setPiece()**
  setPiece()は指定した位置に駒を配置するメソッドである。与えられた座標と駒の種類を board に入れている。
- **int rollPiede()**
  rollPiece()は指定した駒を指定した方向へ転がした場合の駒の種類を返すメソッドである。駒が移動し回転すると種類が変わる。駒の種類を場合分けすることで与えられた駒の種類が移動先でどの駒の種類になるか判断し返している。
- **void showBoard()**
  showBoard()は盤面を表示するメソッドである。６ ×６ の盤面と、そこに配置された駒の上面

および 4 つの側面に描かれたジャンケンの手の種類が表示される。

- **void　movePiece()**
  mobePiece()は指定した位置にある駒を指定した方向へ指定した方法で移動させるメソッドである。
- **void　showPiece()**
  showPiece()は指定した位置にある駒を表示するメソッドである。
- **int　attack()**
  attack()は指定した位置にある駒で攻撃するメソッドである。指定した位置にある駒が勝利したかを返している。

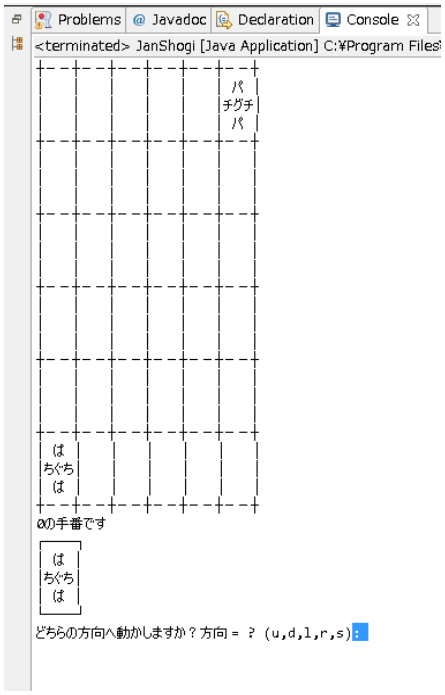
3.3 Constants インタフェース

Constants は定数を定義するインタフェースである。

4　結果および考察

本研究で作成した簡易版ジャンケン将棋のプログラムは、対人戦使用にしておりプレイヤー0とプレイヤー1 の対戦としてある。駒を立体に見せるために駒の上面を真ん中にし、上面から見て上下左右の面をそれぞれ上下左右に表示するようにした。ゲーム盤は 6×6 マスを表示する。自分の手番ではゲーム盤(現状)と「(手番プレイヤー名[0 か 1])の手番です」と今の自駒の現状と自駒の方向選択の「どちらの方向へ動かしますか？方向＝？(u,d,l,r,s)」を表示する。方向は上が u、下が d、左が l、右が r、そのままが s にしてある。方向選択で選択した方向に移動不可能であれば再度選択させる。移動可能であれば、移動時に駒を回転させるか回転させないかの選択に移る。選択時は「転がすかずらすかを選んでください？移動方法＝？(r,s)」と表示する。移動方法は回転するなら r 、そのままなら s にしてある。この後に勝利判定を行う。自駒と相手駒がこの時隣(上下左右のどちらかのマス)あっていれば、互いの上面の手(グー、チョキ、パー)を見て判断する。引き分けなら続行してそのまま行われる。勝ちならゲーム盤(現状)と「(手番プレイヤー名[0 か 1])の勝ち！」と表示し、終了する。負けならゲーム盤(現状)と「(手番プレイヤー名[0 か 1])の負け！」と表示し、終了する。

本研究のプログラムの実行開始画面は図 6 に、実行終了画面は図 7 に示す。



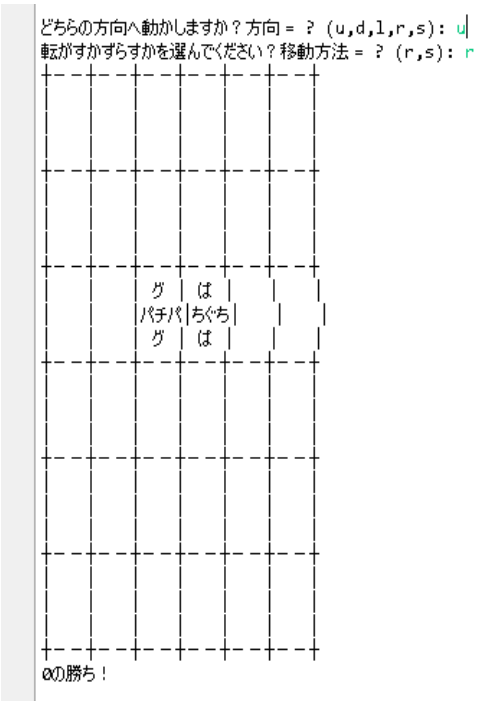


図６　簡易版ジャンケン将棋開始時　　図７　簡易版ジャンケン将棋終了時

結果、以上の事を行えるが、まだ改善点はいくつもある。対人戦だけでＣＰＵ戦ができない点・アプリ化できてない点・本来のジャンケン将棋より簡易化されている点などがある。

対人戦だけでＣＰＵ戦ができない点は開始時に対人戦かＣＰＵ戦を選択できるようにし、ＣＰＵのレベルも選択できるようにする。ＣＰＵのレベルの弱いのはランダムで配置を行うもので強いのはＡＩの作成をする。ＡＩの手法は第１章の１．３ゲームのＡＩで述べた手法の中からより適したものを使用する。

アプリ化できてない点はApplet()クラスを継承したクラスを作成する。そのクラスではユーザインターフェースを作成する。また、ここで配置するボタンなどを作成、マウス入力、画面の表示などをする事で今の現状より良いゲーム環境の作ることができる。

本来のジャンケン将棋より簡易化されている点は、双方のプレイヤーの駒を１個から４個に変更し、初期配置時は上面は選択式にする。手番の行動力を１から２に変更する。勝利条件に相手のゴールに自駒をいれると勝利となることを追加する。駒の移動時は必ず回転する。以上の変更はJanShogi()クラスを変更する必要がある。

5　結論・今後の課題

本研究で作成するジャンケン将棋の簡易版は、双方駒 1 個ずつを用いて対戦できるものを目指している。しかしプログラムが未完成のため、対人戦を行えるように作成していくことが急務である。本研究のプログラムでは双方１個ずつであり、各手番で１種のみという本来のジャンケン将棋より簡易化されている点、対 CPU 戦が行えないという点、アプリ化のためのユーザインターフェースが実装されていない点などといった問題点が多数ある。駒１個、各手番で１種という点は本来のジャンケン将棋のルールに応じたものに改善する必要がある。また、対 CPU 戦を行えるようにジャンケン将棋 AI の作成する必要がある。一般に、将棋やチェス等の AI では、定跡データベースの利用、先読みによる盤面の評価値計算、終盤での詰み読みといった手法が取られる。ジャンケン将棋でも同様に、これらの手法を用いることで AI を作成することが可能であると思われる。また、ユーザインターフェースについては Applet クラスを継承したクラスの導入することにより、ユーザにとってよりプレイし易い環境を構築することが必要である。　完成以降はジャンケン将棋 AI の作成を目指す。ジャンケン将棋 AI の着手選択方法としては以下の点を考慮することが考えられる。まずジャンケン将棋の勝利条件の一つは相手のゴールに入る事があるので、よりゴールに近付く手の評価値を高くする。また相手駒との相対位置と、互いの駒の出目から、駒を転がす方向を決定するためのデータベースを構築する。

またジャンケン将棋には駒の取り合いといった要素もあるので、次の一手で想定される最大の損害を最小になるように選択するミニマックス法も考えられる。

参考文献

[1] Janos Wagner and Istvan Virag, Solving renju, ICGA Journal, Vol.24, No.1, pp.30-35 (2001),
http://www.sze.hu/~gtakacs/download/wagnervirag_2001.pdf

[2] Jonathan Schaeffer, Neil Burch, Yngvi Bjorsson, Akihiro Kishimoto, Martin Muller, Robert Lake, Paul Lu,and Steve Suphen, Checkers is solved, Science Vol.317, No,5844, pp.1518-1522 (2007).http://www.sciencemag.org/content/317/5844/1518.full.pdf

[3] 北尾まどか, 藤田麻衣子, どうぶつしょうぎねっと, (2010), http://dobutsushogi.net/

[4] 田中哲郎, 「どうぶつしょうぎ」の完全解析, 研究報告ゲーム情報学(GI), Vol.2009-GI-22 No.3,
pp.1-8, 情報処理学会, (2009), http://id.nii.ac.jp/1001/00062415/

[5] 清慎一, 川嶋俊, 探索プログラムによる四路盤囲碁の解, 研究報告ゲーム情報学(GI), Vol. 2000-GI-004, pp.69-76, 情報処理学会, (2000), http://id.nii.ac.jp/1001/00058633/

[6] Eric C.D. van der Welf, H.Jaap van den Herik, and Jos W.H.M.Uiterwijk, Solving Go on Small Boards,ICGA Journal, Vol.26, No.2, pp.92-107 (2003).

[7] アンパンマンはじめて将棋, セガトイズ (2012)
http://www.segatoys.co.jp/anpan/product/popup/_legacy/learn/06.htm

[8] 塩田好, 石水隆, 山本博史 :「アンパンマンはじめてしょうぎ」の完全解析, 2013年度 情報処理学会関西支部 支部大会 講演論文集,(2013) , http://id.nii.ac.jp/1001/00096792/

[9] あたまのよくなるゲーム じゃんけんしょうぎ,学研教育出版,(2010)
http://hon.gakken.jp/book/1575033700

付録

以下に本研究で作成したプログラムのソースを示す。

JanShogi()クラス

```
package janshougi;

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


/**
 * ジャンケン将棋(の簡易版)
 */
public class JanShogi implements Constants {
    int turn = 0; // 手番
    Board board;  // ゲーム盤
    final int boardSize = 6;
    int x, y;     // 駒の座標
    boolean autoSelect = true; // 自駒の自動選択

    /**
     * コンストラクタ
     */
    JanShogi() {
        board = new Board();
    }

    /**
     * 手番プレイヤーが1手動かす
     */
    void player() {
        int d;        // 駒を動かす方向 (UP, DOWN, LEFT, RIGHT)
        int m;        // 駒の動かし方 (ROLL, SHIFT);
        String inputString; // 入力文字列
        int newx, newy; // 駒を動かした後の座標

        Scanner keyBoardScanner = new Scanner(System.in);
        System.out.println (turn + "の手番です");

        x=0; y=0;
        selectPiece ();
        board.showPiece (x, y);
        System.out.print ("どちらの方向へ動かしますか？");
        while (true) { // 適切な方向が選択されるまでループ
            System.out.print ("方向 = ? (u,d,l,r,s): ");
            inputString = keyBoardScanner.next();
```

```
    if (inputString.equals ("u")) {
        d = UP;
        if (x==1) {
            System.out.println ("そちらへは動かせません");
            continue;
        } else break;
    } else if (inputString.equals ("d")) {
        d = DOWN;
        if (x==6) {
            System.out.println ("そちらへは動かせません");
            continue;
        } else break;
    } else if (inputString.equals ("l")) {
        d = LEFT;
        if (y==1) {
            System.out.println ("そちらへは動かせません");
            continue;
        } else break;
    } else if (inputString.equals ("r")) {
        d = RIGHT;
        if (y==6) {
            System.out.println ("そちらへは動かせません");
            continue;
        } else break;
    } else if (inputString.equals ("s")) {
        d = STOP;
        break;
    } else {
        System.out.println ("u,d,l,r,s のどれかを入力してください"); continue;
    }
}
if (d != STOP) {
    System.out.print ("転がすかずらすかを選んでください？");
    while (true) { // 適切な移動方法が入力されるまでループ
        System.out.print ("移動方法 = ? (r,s): ");
        inputString = keyBoardScanner.next();
        if (inputString.equals ("r")) {
            m = ROLL;   break;
        } else if (inputString.equals ("s")) {
            m = SHIFT;  break;
        } else {
        System.out.println ("r, s のどちらかを入力してください"); continue;
        }
    }
    board.movePiece (x, y, d, m);
}
board.showBoard();
```

```
    switch (d) {
        case UP:
            newx = x-1;
            newy = y;
            if (newx < 1) newx = 1;
            break;
        case DOWN:
            newx = x+1;
            newy = y;
            if (newx > boardSize) newx = boardSize;
            break;
        case LEFT:
            newx = x;
            newy = y-1;
            if (newy < 1) newy = 1;
            break;
        case RIGHT:
            newx = x;
            newy = y+1;
            if (newy > boardSize) newy = boardSize;
            break;
        default:
            newx = x;
            newy = y;
            break;
    }
    int isWin = board.attack(newx, newy);

    if (isWin == 1) {
        System.out.println (turn + "の勝ち！");
        System.exit(0);
    } else if (isWin == -1) {
        System.out.println (turn + "の負け！");
        System.exit(0);
    }
}

/**
 * 駒を選択する
 */
void selectPiece () {
    if (autoSelect) {
        L: for (int i=1; i<=boardSize; ++i) {
            for (int j=1; j<=boardSize; ++j) {
                switch (board.board[i][j]) {
                    case GPC0:
                    case GCP0:
                    case PGC0:
```

```
                case PCG0:
                case CGP0:
                case CPG0:
                    if (turn == 0) {
                        x=i;
                        y=j;
                        break L;
                    }
                    break;
                case GPC1:
                case GCP1:
                case PGC1:
                case PCG1:
                case CGP1:
                case CPG1:
                    if (turn == 1) {
                        x=i;
                        y=j;
                        break L;
                    }
                    break;
            }
        }
    }
} else {
    Scanner keyBoardScanner = new Scanner(System.in);
    String inputString;
    System.out.println ("動かす駒を選んでください");
    while (true) { // 適切な駒が選択されるまでループ
        while (true) { // 適切なx位置が選択されるまでループ
            System.out.print ("X = ? (1-6) : ");
            inputString = keyBoardScanner.next();
            try {
                x = Integer.parseInt (inputString);
            } catch (NumberFormatException e) { // 整数値以外が入力された場合
                System.out.println ("1～6を入力してください");
                continue;
            }
            if (x<1 || 6<x) {
                System.out.println ("1～6を入力してください");
                continue;
            }
            break;
        }
        while (true) { // 適切なy位置が選択されるまでループ
            System.out.print ("Y = ? (1-6) : ");
            inputString = keyBoardScanner.next();
            try {
```

```
                y = Integer.parseInt (inputString);
            } catch (NumberFormatException e) { // 整数値以外が入力された場合
                System.out.println ("1～6を入力してください");
                continue;
            }
            if (y<1 || 6<y) {
                System.out.println ("1～6を入力してください");
                continue;
            }
            break;
        }
        int piece = board.board[x][y];
        switch (piece) {
            case EMPT:
                System.out.println ("空マスです");
                continue;
            case GPC1:
            case GCP1:
            case PGC1:
            case PCG1:
            case CGP1:
            case CPG1:
                if (turn == 0) {
                    System.out.println ("敵の駒です");
                    continue;
                } else break;
            case GPC0:
            case GCP0:
            case PGC0:
            case PCG0:
            case CGP0:
            case CPG0:
                if (turn == 1) {
                    System.out.println ("敵の駒です");
                    continue;
                } else break;
            default:
                System.out.println ("？？？");
                break;
        }
        break;
    }
  }
}

/**
 * メインメソッド
 */
```

```
    public static void main (String[] args) {
        JanShogi js = new JanShogi();
        js.board.showBoard();
        while (true) {
            js.player();
            if (js.turn == 0) js.turn = 1; else js.turn = 0;
         }

    }
}
```

Board()クラス

```
package janshougi;

/**
 * ジャンケン将棋の盤面を管理するクラス
 */
public class Board implements Constants {
    int board[][];
    final int boardSize = 6;

    /**
     * コンストラクタ
     */
    Board() {
        board = new int[boardSize+2][boardSize+2];

        for (int i=0; i<boardSize; ++i)
            for (int j=0; j<boardSize; ++j)
                board[i][j] = EMPT;
        board[1][boardSize] = GPC1;
        board[boardSize][1] = GPC0;
    }

    /**
     * 指定したマスに駒を配置する
     * @param int p : 駒の種類
     * @param int x : x座標
     * @param int y : y座標
     */
    public void setPiece (int p, int x, int y) {
        board[x][y] = p;
    }

    /**
     * 指定した駒を指定した方向へ転がした場合の駒の種類
     * @param int p : 駒の種類
```

```
 * @paran int d : 転がす方向
 * @return int  : 転がした後の駒の種類
 */
public int rollPiece (int p, int d) {
    switch (p) {
        case EMPT : return EMPT;
        case GPC0 :
            switch (d) {
                case UP    : return PGC0;
                case DOWN  : return PGC0;
                case RIGHT : return CPG0;
                case LEFT  : return CPG0;
                default    : return ERR;
            }
        case GCP0 :
            switch (d) {
                case UP    : return GCP0;
                case DOWN  : return GCP0;
                case RIGHT : return PCG0;
                case LEFT  : return PCG0;
                default    : return ERR;
            }
        case PGC0 :
            switch (d) {
                case UP    : return GPC0;
                case DOWN  : return GPC0;
                case RIGHT : return CGP0;
                case LEFT  : return CGP0;
                default    : return ERR;
            }
        case PCG0 :
            switch (d) {
                case UP    : return CPG0;
                case DOWN  : return CPG0;
                case RIGHT : return GCP0;
                case LEFT  : return GCP0;
                default    : return ERR;
            }
        case CGP0 :
            switch (d) {
                case UP    : return GCP0;
                case DOWN  : return GCP0;
                case RIGHT : return PGC0;
                case LEFT  : return PGC0;
                default    : return ERR;
            }
        case CPG0 :
            switch (d) {
```

```
            case UP    : return GCP0;
            case DOWN  : return GCP0;
            case RIGHT : return PGC0;
            case LEFT  : return PGC0;
            default    : return ERR;
        }
    case GPC1 :
        switch (d) {
            case UP    : return PGC1;
            case DOWN  : return PGC1;
            case RIGHT : return CPG1;
            case LEFT  : return CPG1;
            default    : return ERR;
        }
    case GCP1 :
        switch (d) {
            case UP    : return CGP1;
            case DOWN  : return CGP1;
            case RIGHT : return PCG1;
            case LEFT  : return PCG1;
            default    : return ERR;
        }
    case PGC1 :
        switch (d) {
            case UP    : return GPC1;
            case DOWN  : return GPC1;
            case RIGHT : return CGP1;
            case LEFT  : return CGP1;
            default    : return ERR;
        }
    case PCG1 :
        switch (d) {
            case UP    : return CPG1;
            case DOWN  : return CPG1;
            case RIGHT : return GCP1;
            case LEFT  : return GCP1;
            default    : return ERR;
        }
    case CGP1 :
        switch (d) {
            case UP    : return GCP1;
            case DOWN  : return GCP1;
            case RIGHT : return PGC1;
            case LEFT  : return PGC1;
            default    : return ERR;
        }
    case CPG1 :
        switch (d) {
```

```
                case UP    : return GCP1;
                case DOWN  : return GCP1;
                case RIGHT : return PGC1;
                case LEFT  : return PGC1;
                default    : return ERR;
            }
        default : return ERR;
    }
}

/**
 * 盤面を表示する
 */
public void showBoard() {
    for (int j=1; j<=boardSize; ++j)
        System.out.print ("┼－－");
    System.out.println ("┼");
    for (int i=1; i<=boardSize; ++i) {
        System.out.print ("|");
        for (int j=1; j<=boardSize; ++j) {
            switch (board[i][j]) {
                case EMPT: System.out.print ("      "); break;
                case GPC0: System.out.print ("  ぱ  "); break;
                case GCP0: System.out.print ("  ち  "); break;
                case PGC0: System.out.print ("  ぐ  "); break;
                case PCG0: System.out.print ("  ち  "); break;
                case CGP0: System.out.print ("  ぐ  "); break;
                case CPG0: System.out.print ("  ぱ  "); break;
                case GPC1: System.out.print ("  パ  "); break;
                case GCP1: System.out.print ("  チ  "); break;
                case PGC1: System.out.print ("  グ  "); break;
                case PCG1: System.out.print ("  チ  "); break;
                case CGP1: System.out.print ("  グ  "); break;
                case CPG1: System.out.print ("  パ  "); break;
                default  : System.out.print ("  ？  "); break;
            }
            System.out.print ("|");
        }
        System.out.println();
        System.out.print ("|");
        for (int j=1; j<=boardSize; ++j) {
            switch (board[i][j]) {
                case EMPT: System.out.print ("      "); break;
                case GPC0: System.out.print ("ちぐち"); break;
                case GCP0: System.out.print ("ぱぐぱ"); break;
                case PGC0: System.out.print ("ちぱち"); break;
                case PCG0: System.out.print ("ぐぱぐ"); break;
                case CGP0: System.out.print ("ぱちぱ"); break;
```

```
                case CPG0: System.out.print ("ぐちぐ"); break;
                case GPC1: System.out.print ("チグチ"); break;
                case GCP1: System.out.print ("パグパ"); break;
                case PGC1: System.out.print ("チパチ"); break;
                case PCG1: System.out.print ("グパク"); break;
                case CGP1: System.out.print ("パチパ"); break;
                case CPG1: System.out.print ("グチグ"); break;
                default  : System.out.print ("？？？"); break;
            }
            System.out.print ("|");
        }
        System.out.println();
        System.out.print ("|");
        for (int j=1; j<=boardSize; ++j) {
            switch (board[i][j]) {
                case EMPT: System.out.print ("     "); break;
                case GPC0: System.out.print ("  ぱ  "); break;
                case GCP0: System.out.print ("  ち  "); break;
                case PGC0: System.out.print ("  ぐ  "); break;
                case PCG0: System.out.print ("  ち  "); break;
                case CGP0: System.out.print ("  ぐ  "); break;
                case CPG0: System.out.print ("  ぱ  "); break;
                case GPC1: System.out.print ("  パ  "); break;
                case GCP1: System.out.print ("  チ  "); break;
                case PGC1: System.out.print ("  グ  "); break;
                case PCG1: System.out.print ("  チ  "); break;
                case CGP1: System.out.print ("  グ  "); break;
                case CPG1: System.out.print ("  パ  "); break;
                default  : System.out.print ("  ？  "); break;
            }
            System.out.print ("|");
        }
        System.out.println();
        for (int j=1; j<=boardSize; ++j)
            System.out.print ("┼－－");
        System.out.println ("┼");
    }

}

/**
 * 指定したマスにある駒を指定した方向へ指定した方法で移動させる
 * @param int x : x座標
 * @param int y : y座標
 * @paran int d : 方向 (UP, DOWN, LEFT, RIGHT)
 * @param int m : 違法方法 (ROLL, SHIFT)
 */
public void movePiece (int x, int y, int d, int t) {
```

```
    int newx, newy;
    int piece = board[x][y];
    switch (d) {
        case UP:
            newx = x-1;
            newy = y;
            if (newx < 1) newx = 1;
            break;
        case DOWN:
            newx = x+1;
            newy = y;
            if (newx > boardSize) newx = boardSize;
            break;
        case LEFT:
            newx = x;
            newy = y-1;
            if (newy < 1) newy = 1;
            break;
        case RIGHT:
            newx = x;
            newy = y+1;
            if (newy > boardSize) newy = boardSize;
            break;
        default:
            newx = x;
            newy = y;
            break;
    }
    if (t == SHIFT) {
        board[x][y] = EMPT;
        board[newx][newy] = piece;
    } else if (t == ROLL) {
        board[x][y] = EMPT;
        board[newx][newy] = rollPiece (piece, d);
    }
}

/**
 * 指定したマスにある駒を表示する
 */
public void showPiece (int x, int y) {
    System.out.println ("┌──┐");
    System.out.print ("│");
    switch (board[x][y]) {
        case EMPT: System.out.print ("    "); break;
        case GPC0: System.out.print (" ぱ "); break;
        case GCP0: System.out.print (" ち "); break;
        case PGC0: System.out.print (" ぐ "); break;
```

```
    case PCG0: System.out.print ("  ち  "); break;
    case CGP0: System.out.print ("  ぐ  "); break;
    case CPG0: System.out.print ("  ぱ  "); break;
    case GPC1: System.out.print ("  パ  "); break;
    case GCP1: System.out.print ("  チ  "); break;
    case PGC1: System.out.print ("  グ  "); break;
    case PCG1: System.out.print ("  チ  "); break;
    case CGP1: System.out.print ("  グ  "); break;
    case CPG1: System.out.print ("  パ  "); break;
    default  : System.out.print ("  ？  "); break;
}
System.out.println ("|");
System.out.print ("|");
switch (board[x][y]) {
    case EMPT: System.out.print ("      "); break;
    case GPC0: System.out.print ("ちぐち"); break;
    case GCP0: System.out.print ("ぱぐぱ"); break;
    case PGC0: System.out.print ("ちぱち"); break;
    case PCG0: System.out.print ("ぐぱぐ"); break;
    case CGP0: System.out.print ("ぱちぱ"); break;
    case CPG0: System.out.print ("ぐちぐ"); break;
    case GPC1: System.out.print ("チグチ"); break;
    case GCP1: System.out.print ("パグパ"); break;
    case PGC1: System.out.print ("チパチ"); break;
    case PCG1: System.out.print ("グパグ"); break;
    case CGP1: System.out.print ("パチパ"); break;
    case CPG1: System.out.print ("グチグ"); break;
    default  : System.out.print ("？？？"); break;
}
System.out.println ("|");
System.out.print ("|");
switch (board[x][y]) {
    case EMPT: System.out.print ("      "); break;
    case GPC0: System.out.print ("  ぱ  "); break;
    case GCP0: System.out.print ("  ち  "); break;
    case PGC0: System.out.print ("  ぐ  "); break;
    case PCG0: System.out.print ("  ち  "); break;
    case CGP0: System.out.print ("  ぐ  "); break;
    case CPG0: System.out.print ("  ぱ  "); break;
    case GPC1: System.out.print ("  パ  "); break;
    case GCP1: System.out.print ("  チ  "); break;
    case PGC1: System.out.print ("  グ  "); break;
    case PCG1: System.out.print ("  チ  "); break;
    case CGP1: System.out.print ("  グ  "); break;
    case CPG1: System.out.print ("  パ  "); break;
    default  : System.out.print ("  ？  "); break;
}
System.out.println ("|");
```

```
        System.out.println ("└─────┘");
    }

    /**
     * 指定した位置にある駒で攻撃
     * @param int x : x座標
     * @param int y : y座標
     * @return int : 指定した位置にある駒が勝利したか？ (1:勝 -1:負 0:分)
     */
    public int attack (int x, int y) {
        int selfPiece = board[x][y];
        int enemyPiece = EMPT;

        if (selfPiece == EMPT) return 0;

        if (board[x-1][y] != EMPT) enemyPiece = board[x-1][y];
        else if (board[x+1][y] != EMPT) enemyPiece = board[x+1][y];
        else if (board[x][y-1] != EMPT) enemyPiece = board[x][y-1];
        else if (board[x][y+1] != EMPT) enemyPiece = board[x][y+1];

        if (enemyPiece == EMPT) return 0;

        switch (selfPiece) {
            case GPC0:
            case GCP0:
                switch (enemyPiece) {
                    case GPC1:
                    case GCP1:
                        return 0;
                    case PGC1:
                    case PCG1:
                        return -1;
                    case CGP1:
                    case CPG1:
                        return 1;
                    default :
                        return 0;
                }
            case PGC0:
            case PCG0:
                switch (enemyPiece) {
                    case GPC1:
                    case GCP1:
                        return 1;
                    case PGC1:
                    case PCG1:
                        return 0;
                    case CGP1:
```

```
            case CPG1:
                return -1;
            default :
                return 0;
        }
    case CGP0:
    case CPG0:
        switch (enemyPiece) {
            case GPC1:
            case GCP1:
                return -1;
            case PGC1:
            case PCG1:
                return 1;
            case CGP1:
            case CPG1:
                return 0;
            default :
                return 0;
        }
    case GPC1:
    case GCP1:
        switch (enemyPiece) {
            case GPC0:
            case GCP0:
                return 0;
            case PGC0:
            case PCG0:
                return -1;
            case CGP0:
            case CPG0:
                return 1;
            default :
                return 0;
        }
    case PGC1:
    case PCG1:
        switch (enemyPiece) {
            case GPC0:
            case GCP0:
                return 1;
            case PGC0:
            case PCG0:
                return 0;
            case CGP0:
            case CPG0:
                return -1;
            default :
```

```
                    return 0;
                }
            case CGP1:
            case CPG1:
                switch (enemyPiece) {
                    case GPC0:
                    case GCP0:
                        return -1;
                    case PGC0:
                    case PCG0:
                        return 1;
                    case CGP0:
                    case CPG0:
                        return 0;
                    default :
                        return 0;
                }
            default:
                return 0;
        }
    }

    /**
     * テスト用メインメソッド
     */
    public static void main (String args[]) {
        Board b = new Board();
        b.setPiece (GPC0, 6, 1);
        b.setPiece (GCP0, 1, 2);
        b.setPiece (PGC0, 2, 3);
        b.setPiece (PCG0, 3, 4);
        b.setPiece (CGP0, 4, 5);
        b.setPiece (CPG0, 5, 6);
        b.setPiece (GPC1, 6, 3);
        b.setPiece (GCP1, 1, 4);
        b.setPiece (PGC1, 2, 5);
        b.setPiece (PCG1, 3, 6);
        b.setPiece (CGP1, 4, 1);
        b.setPiece (CPG1, 5, 2);
        b.showBoard();
    }
}
```

Constants()クラス

```
package janshougi;

/**
```

```
 * 定数を定義するインタフェース
 */
public interface Constants {
    public static final int UP = 1;
    public static final int DOWN = 2;
    public static final int RIGHT = 3;
    public static final int LEFT = 4;
    public static final int STOP = 0;

    public static final int EMPT = 0;
    /* 空白*/
    public static final int GPC0 = -1;
    /* 先手の駒
              P
      GPC0 = CGC
              P
    */
    public static final int GCP0 = -2;
    /* 先手の駒
              C
      GCP0 = PGP
              C
    */
    public static final int PGC0 = -3;
    /* 先手の駒
              G
      GPC0 = CPC
              G
    */
    public static final int PCG0 = -4;
    /* 先手の駒
              C
      GCP0 = GPG
              C
    */
    public static final int CGP0 = -5;
    /* 先手の駒
              G
      CGP0 = PCP
              G
    */
    public static final int CPG0 = -6;
    /* 先手の駒
              P
      CPG0 = GCG
              P
    */
    public static final int GPC1 = 1;
```

```
    /* 後手の駒
              P
      GPC1 = CGC
              P
    */
    public static final int GCP1 = 2;
    /* 後手の駒
              C
      GCP1 = PGP
              C
    */
    public static final int PGC1 = 3;
    /* 後手の駒
              G
      GPC1 = CPC
              G
    */
    public static final int PCG1 = 4;
    /* 後手の駒
              C
      GCP1 = GPG
              C
    */
    public static final int CGP1 = 5;
    /* 後手の駒
              G
      CGP1 = PCP
              G
    */
    public static final int CPG1 = 6;
    /* 後手の駒
              P
      CPG1 = GCG
              P
    */

    public static final int ERR = Integer.MAX_VALUE;
    /* エラー */

    public static final int ROLL = 0;
    /* 転がす */
    public static final int SHIFT = 1;
    /* ずらす */
}
```