卒業研究報告書

題目

虱潰し探索により詰将棋を解くプログラム

指導教員

石水　隆　講師

報告者

13-1-037-0179

玉置　康平

近畿大学理工学部情報学科

平成 29 年 1 月 31 日提出

概要

近年、電王戦などコンピュータ将棋の開発が進み、注目を集めている。今回はその中でも、詰将棋の部分に注目した。

詰将棋は本来、本将棋の終盤力を鍛えるための練習問題という位置付けであったが、現在ではパズルの一種として広く親しまれている。

詰将棋は、大駒を捨てたり飛角歩を不成りにしたりなど、本将棋とは異なる手順がよく見受けられる．このため，コンピュータ将棋では終盤は通常の着手選択アルゴリズムとは別に，詰将棋アルゴリズムを走らせて詰みがある場合に詰みを逃さないようにしているものも多い．

本研究は局面を与えると虱潰し探索を用いて全ての王手のパターンを探索し、詰みの有無の判断を行い詰将棋を解くプログラムの開発を目的としている。

目次

1　序論

1.1 本研究の背景

詰将棋とは、与えられた局面から先手が王手の連続で玉を詰みの状態にする事を目的にする問題である。詰め将棋はその性質上、ゲーム木の探索問題として定式化でき、昔から詰め将棋を解くプログラムは研究され続けていた。

多くのコンピュータ将棋では，着手選択の際，駒得できる手や有利な位置を確保できる手を高評価詰とし、無駄に駒を捨てたりしないようにしている。一方、詰将棋は、大駒を捨てたり飛角歩を不成りにしたりなど、本将棋とは異なる手順がよく見受けられる．このため，コンピュータ将棋では終盤は通常の着手選択アルゴリズムとは別に，詰将棋アルゴリズムを走らせて詰みがある場合に詰みを逃さないようにしているものも多い．

詰め将棋の探索方法は大まかに、虱潰し探索法と最優良探索法の二種類に分けられる。前者は手数が長くなるにつれて探索範囲が爆発的に増えるため、長手数の詰め将棋には不向きである。一方、後者は長手数の詰め将棋を解くのに有望であるが、記憶領域を大量に必要とするというデメリットもある。

古来より多くの詰め将棋が作られているが、その多くは短手数の詰将棋であり、棋力の向上にも短手数の詰将棋を多く解くのが良いとされる。そこで本研究では短手数の詰将棋を解ける虱潰し探索を用いた詰将棋を解くプログラムを作成し、その有用性を検証する。

1.2 詰将棋を解くプログラムの既知の結果

詰将棋とコンピュータの歴史は長く、1970 年頃にはプログラムの開発もなされていた[３]。当時は深さ優先探索が主流であり、25 手詰め程度が実用的な範囲の物であったと言われている。研究が進むにつれ、最優良探索法が採用されることが多くなり、現在ではそのどちらも取り入れたハイブリッド探索法が主流である。

詰将棋を解くプログラムの進化は早く、「脊尾詰」というプログラムが登場し、1995 年には 873 手詰めの「新扇詰」 を解いている。そして 1997 年には現在の最長手数の 1925 手詰めの詰将棋「ミクロコスモス」を解くという偉業をなしとげている[1]。

また、Ito,T2 といったプログラムが、プロでも全問解くのに 35 分はかかると言われる問題集をそれぞれ２分半、１分という驚異的な早さで解いている[2]。

これらのことから、コンピュータによる詰将棋を解くということに関しては、既に人間が解くよりも深さ、速さ共に大きく上回っていると言える。

1.3 本研究の目的

本研究の目的は虱潰し探索により詰将棋を解くプログラムを開発し、その有用性を確かめるものである。現在、著名な将棋ソフトのほとんどが最優良探索を用いているが、記憶領域を多量に消費してしまうなどのデメリットがある。そこで本研究ではそのデメリットを回避するため、虱潰し探索を採用した。

1.4 本報告書の構成

本報告書の構成は以下の通りである。

２章では、本プログラムで使用する盤駒の種類や詰将棋のルールなどの前提条件について述べる。

３章では、作成したアルゴリズムについて述べる。

４章では、結果と考察について述べる。

2　詰将棋のルール

本章では詰将棋について述べる。

詰め将棋とは、与えられた局面から先手は常に王手を掛け続け、後手は常に王手を回避する様に交互に指し続け、玉を詰みの状態にする事を目的にする問題である。

詰みの状態とは、後手側がどんな手をさしても次の先手番で玉を取られる様な局面のことを指す。後手が最善の手を指し続けても詰みになる場合、詰将棋として確立する。

詰将棋には以下に挙げるルールがある[14]。

- 王手の連続で玉を詰めること
- 攻め方は最短手順で玉を詰める
- 玉方は最長手順を選ぶこと
- 玉方は盤上の駒と攻め方の持ち駒以外の駒を使ってよい
- 攻め方は持ち駒と、王手をしながら取った駒を使ってよい
- 玉方は逃げ手順で、同手数の二つの手順がある場合、攻め方に駒を与えない方を正解とする

その他は指し将棋と同じである。二歩、打ち歩詰め、行き所の無い駒を打つことは禁手である。また連続王手は千日手ではなく、失敗である。

また、詰将棋では、攻め方が手を変えたときに詰みとなる手順が複数ある場合、または玉方が手を変えたときに同手数でかつ持ち駒を使い切って詰みとなる手順が複数ある場合、余詰めとして詰将棋としての価値が低いとみなされる。しかし、本将棋においては余詰めがあっても最終的に相手玉を詰めることができれば充分である。よって、本研究では、余詰めも含めて詰ます手順を求める。

3 詰将棋を解くプログラムで用いた手法

詰将棋を解く手法は大まかに最優良探索と虱潰し探索に分けられる。

最新の将棋ソフトのほとんどが最優良探索を用いているが、虱潰し探索は有力では無いのか調べるために、今回は後者を採用している。

以下本研究で用いた手法について述べる

3.1 王手の判定

王手の判定は、与えられた局面において、自分の駒の可動範囲を全て確認して、その中に相手玉が含まれていた場合、王手とみなす。

3.2 詰みの判定

詰みの判定は、王手と判定された状態で、相手玉の駒の可動範囲を全て手確認しても王手の判定が続いている場合、詰みとみなす。

しかし、2.2.2 で述べた様な反則手の場合、詰みとはみなさない。

3.3 着手の選択

先手番は自分の駒が取りうる全ての動きを調べ、その中に王手の判定を発生するものがあればその手を選択する。

後手番は自分の駒が取りうる全ての動きを調べ、その中に王手の判定を回避するものがあればその手を選択する。

4 結果と考察

本研究で作成したプログラムの検証をするため、いくつかの詰将棋を与えて実行させた。例として、図 1 の局面を与えた場合を示す。正解手順は▲２四龍,』 同金, ▲3 三龍の三手詰めである。この局面をプログラムに解かせたところ、図 2 に示す結果となった。正解手順が示されており、詰め将棋の解答が出来ているといえる。

また、n 手詰めの問題を解くとき、最優良探索と虱潰し探索との空間計算量の差は分枝率をｂと置くと、表１のようになる。

表 1 各探索の空間計算量

探索法	空間計算量	時間計算量
最優良探索	$O(b^n)$	$O(b^n)$ (最悪時)
虱潰し深さ優先探索	O(n)	$O(b^n)$

表１に示される通り、時間計算量については最悪時はどちらも b^n であるが、最優良探索は高性能な評価を行えば、計算量を減少させることが出来る。

空間計算量については、深さ優先の虱潰し探索は線形であり、常に ｎになるが、最優良探索の空間計算量は指数関数的に増えるため、記憶領域を大量に消費してしまう。

よって虱潰し深さ優先探索では計算時間はかかるものの、記憶領域の消費が少ないことが確認できる。

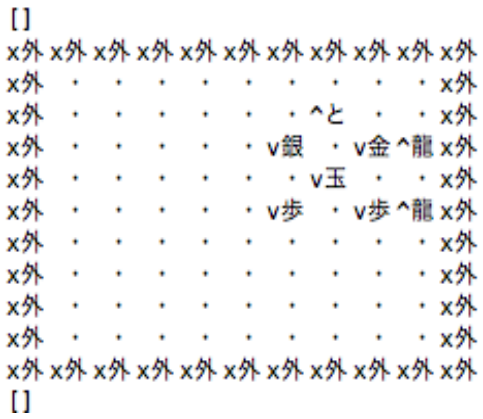

図 1　出題図

図 1　詰め上がり図

5　結論・今後の課題

　本研究の目的である虱潰し探索を用いた詰将棋を解くプログラムの開発は概ね達成したといえる。
　また、本研究で作成したプログラムの応用として、自作した詰将棋が本当に詰むのか、また、余詰めが無いかなど調べることができると考えられる。
今後の課題としては、kif ファイル形式に対応することで、将棋ウォーズや将棋倶楽部２４などで実際に対局した棋譜を読み込み、人間では詰みの判断が難しい局面での最善手を示すことが出来る。
　また、虱潰し探索という性質上、探索木が大きくなるにつれて、解を出し切れなくなるなどの不具合が発生したため、深さに制限を加えることになってしまった。
具体的な対策としては、ある程度の評価関数を設けて、明らかに無駄な探索を省き、有力な順から探索するなど、ある程度最優良探索法を取り入れたハイブリッドな探索法を用いることなどが挙げられる。

謝辞

最後まで不甲斐ない私に真摯にご教授して頂いた石水先生には本当にお世話になりました。
ここに感謝の意を表します。

参考文献

[1] 長井歩,今井浩. df-pn アルゴリズムの詰将棋を解くプログラムへの応用 , 情報処理学会論文誌, Vol.43, No.6, pp.1769-1777, 2002, http://id.nii.ac.jp/1001/00011597/ .

[1] 伊藤琢巳,野下浩平. 詰将棋を早く解く 2 つのプログラムとその評価 , 情報処理学会論文誌, Vol.35, No.8,pp.1531-1539, 1994 , http://id.nii.ac.jp/1001/00014156/ .

[1] 伊藤琢巳,ハイブリット探索法を用いた詰将棋を解くプログラム, 情報処理学会 第 48 回全国大会講演論文集(人工知能及び認知科学), pp.125-126, 1996 , http://id.nii.ac.jp/1001/00125421/

[1] 池泰弘, Java 将棋のアルゴリズム,工学社, 2007

[1] 池泰弘.コンピュータ将棋のアルゴリズム, 工学社, ~~2007~~2005

[2] 松原仁,コンピュータ将棋の進歩,共立出版, 1993

[3] 瀧澤武信, 松原仁, 古作登 著, 人間に勝つコンピュータ将棋の作り方,技術評論社, 2012

[4] 日本将棋連盟, https://www.shogi.or.jp

[5] 将棋日本シリーズ JT プロ公式戦, JT, https://www.jti.co.jp/knowledge/shogi/index.html

[6] やまぐちしげお, 詰将棋の歴史 , 楽しい詰将棋の世界, http://www.asahi-net.or.jp/~xr9s-ymgc/tume/rekisi.html

[7] 池泰弘, コンピュータ将棋の作り方 「うさぴょん」「ねこにゃ」開発日記 http://usapyon.cocolog-nifty.com/shogi/

[8] Bonanza - The Computer Shogi Program, http://www.geocities.jp/bonanza_shogi/

[9] やねうらお, やねうら王 http://yaneuraou.yaneu.com

[10] CSA 標準棋譜ファイル形式, CSA 通信プロトコル, コンピュータ将棋協会, http://www.computer-shogi.org/protocol/record_v22.html

[11] 原田泰夫 著, 強くなる新詰め将棋 200 題, 梧桐書院, 1990.

付録

本研究で作成した詰将棋プログラムのソースを以下に示す。

「Board クラス」

```
package sotuken;

import java.util.ArrayList;
import java.util.Arrays;

/*
 * Board クラス
 * 問題局面を与えて、得られた回答を表示するクラス
 */
public class Board {

	// 自分の駒
	static String p = "^歩";
	static String l = "^香";
	static String n = "^桂";
	static String s = "^銀";
	static String g = "^金";
	static String k = "^玉";
	static String r = "^飛";
```

```
static String x = "x 外"; // 外マス

static ArrayList<String> arrayQ = new ArrayList<String>();

// 問題図
static String question[][] = { // 1 2 3 4 5 6 7 8 9
                        { x, x, x, x, x, x, x, x, x, x, x }, { x, o, o, o, o, o, o, o, en, ek, x }, // 1
                        { x, o, o, o, o, o, o, p, o, ep, x }, // 2
                        { x, o, o, o, o, tb, o, eb, g, o, x }, // 3
                        { x, o, o, o, o, o, o, o, o, o, x }, // 4
                        { x, o, o, o, o, o, o, o, o, n, x }, // 5
                        { x, o, o, o, o, o, o, o, o, o, x }, // 6
                        { x, o, o, o, o, o, o, o, o, o, x }, // 7
                        { x, o, o, o, o, o, o, o, o, o, x }, // 8
                        { x, o, o, o, o, o, o, o, o, o, x }, // 9
                        { x, x, x, x, x, x, x, x, x, x, x } };

static ArrayList<String> motiSente = new ArrayList<String>(Arrays.asList());// 問題局面の先手の持ち駒
static ArrayList<String> motiGote = new ArrayList<String>(Arrays.asList());// 問題局面の後手の持ち駒

/*
 * changeArray メソッド 二次元配列を arrayList に変換する
 */
static void changeArray(String[][] q) {

        for (int z = 0; z < 11; z++) {
                for (int y = 0; y < 11; y++) {
                        arrayQ.add(q[z][y]);
                }
        }
}

/*
 * show メソッド 局面(盤上、先手持ち駒、後手持ち駒)を表示する
 */
static void show(ArrayList<String> q, ArrayList<String> s, ArrayList<String> g) {

        System.out.print(g);// 後手の持ち駒

        System.out.println();

        int count = 0;
        for (int z = 0; z < 11; z++) {
                for (int y = 0; y < 11; y++) {

                        System.out.print(q.get(count));
                        count++;
                }
                System.out.println();

        }

        System.out.print(s);// 先手の持ち駒

        System.out.println();
}
```

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

		ArrayList<kyokumen> answerList = new ArrayList<kyokumen>();
		System.out.println();
		System.out.println("'問題図'");
		changeArray(question);
		show(arrayQ, motiSente, motiGote);
		System.out.println("-----------------------------------------");

		System.out.println("'答え'");
		System.out.println();

		Solve s = new Solve();

		answerList = s.solve(arrayQ, motiSente, motiGote);

		if (answerList.isEmpty()) { // 詰みが無いとき
			System.out.println("解無し");
		} else { // 詰み形を全て表示
			for (int a = 0; a < answerList.size(); a++) {
				show(answerList.get(a).qq, answerList.get(a).motiS,
answerList.get(a).motiG);

			}
		}

	}

}
```

「Koma クラス」

```
package sotuken;

import java.util.ArrayList;
import java.util.Arrays;
/*
 * Koma クラス
 * 駒の動きを表すクラス
 *
 */

public class Koma {

	/*
	 * 駒の可動範囲は数字で表すと以下のようになる
	 *
	 * +-----+-----+-----+
	 * | -20 | -22 | -21 |
	 * +-----+-----+-----+
	 * | -12 | -11 | -10 |
	 * +-----+-----+-----+
```

```
     * | -1 | 駒   | 1 |
     * +-----+-----+-----+
     * | 10 | 11 | 12 |
     * +-----+-----+-----+
     *
     *
     */

    int pawn[] = { -11 }; //歩
    int lance[] = { -11, -22, -33, -44, -55, -66, -77, -88, -99 }; //香車
    int knight[] = { -23, -21 }; //桂馬
    int silver[] = { -12, -11, -10, 10, 12 };//銀
    int gold[] = { -12, -11, -10, -1, 1, 11 }; //金
    int king[] = { -12, -11, -10, 1, -1, 10, 11, 12 }; //玉
    int rook[] = { -11, -22, -33, -44, -55, -66, -77, -88, -99, -1, -2, -3, -4, -5, -6, -7, -8, -9, 1, 2, 3, 4, 5, 6, 7,
                    8, 9, 11, 22, 33, 44, 55, 66, 77, 88, 99 }; //飛車
    int bishop[] = { -12, -24, -36, -48, -60, -72, -84, -96, -108, -10, -20, -30, -40, -50, -60, -70, -80, -90, 10, 20,
                    30, 40, 50, 60, 70, 80, 90, 12, 24, 36, 48, 60, 72, 84, 96, 108 }; //角
    int turn_pawn[] = { -12, -11, -10, -1, 1, 11 }; //と金
    int turn_lance[] = { -12, -11, -10, -1, 1, 11 }; //成香
    int turn_knight[] = { -12, -11, -10, -1, 1, 11 }; //成圭
    int turn_silver[] = { -12, -11, -10, -1, 1, 11 }; //成銀
    int turn_rook[] = { -11, -22, -33, -44, -55, -66, -77, -88, -99, -1, -2, -3, -4, -5, -6, -7, -8, -9, 1, 2, 3, 4, 5,
                    6, 7, 8, 9, 11, 22, 33, 44, 55, 66, 77, 88, 99, -12, -10, 10, 12 }; //龍
    int turn_bishop[] = { -12, -24, -36, -48, -60, -72, -84, -96, -108, -10, -20, -30, -40, -50, -60, -70, -80, -90, 10,
                    20, 30, 40, 50, 60, 70, 80, 90, 12, 24, 36, 48, 60, 72, 84, 96, 108, -11, -1, 1, 11, }; //馬

    //駒の可動範囲をまとめたもの
    int[] komaK[] = { pawn, lance, knight, silver, gold, king, rook, bishop, turn_pawn, turn_lance, turn_knight,
                    turn_silver, turn_rook, turn_bishop };

    //先手の駒の種類
    ArrayList<String> myKomaList = new ArrayList<>(
                    Arrays.asList("^歩", "^香", "^桂", "^銀", "^金", "^玉", "^飛", "^角", "^と", "^杏", "^圭
", "^全", "^龍", "^馬"));
    //後手の駒の種類
    ArrayList<String> enemyKomaList = new ArrayList<>(
                    Arrays.asList("v 歩", "v 香", "v 桂", "v 銀", "v 金", "v 玉", "v 飛", "v 角", "v と", "v 杏
", "v 圭", "v 全", "v 龍", "v 馬"));

    //盤の符号を表す
    ArrayList<Integer> ban = new ArrayList<Integer>(Arrays.asList(
                    0,  0,  0,  0,  0,  0,  0,  0,  0,  0, 0,
                    0, 91, 81, 71, 61, 51, 41, 31, 21, 11, 0,
                    0, 92, 82, 72, 62, 52, 42, 32, 22, 21, 0,
                    0, 93, 83, 73, 63, 53, 43, 33, 23, 13, 0,
                    0, 94, 84, 74, 64, 54, 44, 34, 24, 14, 0,
                    0, 95, 85, 75, 65, 55, 45, 35, 25, 15, 0,
                    0, 96, 86, 76, 66, 56, 46, 36, 26, 16, 0,
                    0, 97, 87, 77, 67, 57, 47, 37, 27, 17, 0,
                    0, 98, 88, 78, 68, 58, 48, 38, 28, 18, 0,
                    0, 99, 89, 79, 69, 59, 49, 39, 29, 19, 0,
                    0,  0,  0,  0,  0,  0,  0,  0,  0,  0, 0));
```

```
/*
 * それぞれの駒が王手をかけているか判定するメソッド
 * @ return true なら王手をかけている
 */
boolean checkP(ArrayList<String> q, int coo) {  //歩

		boolean boo = false;
		for (int c = 0; c < pawn.length; c++) {
			if ((coo + pawn[c]) == q.indexOf("v 玉")) {
				boo = true;
				break;
			}

		}
		return boo;

}

boolean checkL(ArrayList<String> q, int coo) { //香車

		boolean boo = false;

		for (int c = 0; c < lance.length; c++) {

			if ((coo + lance[c]) == q.indexOf("v 玉")) {
				boo = true;
				int dis = ((coo - q.indexOf("v 玉")) / 11); //玉までの距離

				for (int n = 1; n < dis; n++) {
					if (!(q.get(coo - (11 * n)).contains("・ "))) { //玉までに空マスが
無かったとき false を返す

						boo = false;
					}
				}

				break;
			}

		}
		return boo;
}

boolean checkN(ArrayList<String> q, int coo) { //桂馬
		boolean boo = false;

		for (int c = 0; c < knight.length; c++) {
			if ((coo + knight[c]) == q.indexOf("v 玉")) {
				boo = true;

				break;
			}
		}
		return boo;
```

```
}

boolean checkS(ArrayList<String> q, int coo) { //銀
	boolean boo = false;

	for (int c = 0; c < silver.length; c++) {
		if ((coo + silver[c]) == q.indexOf("v 玉")) {
			boo = true;
			break;
		}
	}
	return boo;
}

boolean checkG(ArrayList<String> q, int coo) { //金
	boolean boo = false;

	for (int c = 0; c < gold.length; c++) {
		if ((coo + gold[c]) == q.indexOf("v 玉")) {
			boo = true;
			break;
		}
	}

	return boo;

}

boolean checkK(ArrayList<String> q, int coo) { //玉
	boolean boo = false;

	for (int c = 0; c < gold.length; c++) {
		if ((coo + gold[c]) == q.indexOf("v 玉")) {
			boo = true;
			break;
		}
	}

	return boo;

}

boolean checkR(ArrayList<String> q, int coo) { //飛車
	boolean boo = false;

	for (int c = 0; c < rook.length; c++) {
		if ((coo + rook[c]) == q.indexOf("v 玉")) {
			boo = true;

			if ((coo - q.indexOf("v 玉")) > 11) {// 玉が上のとき

				int dis = ((coo - q.indexOf("v 玉")) / 11);//玉までの距離

				for (int n = 1; n < dis; n++) {
```

```
                                        if (!(q.get(coo - (11 * n)).contains("・ "))) {//玉までに空
マスが無かったとき false を返す

                                                boo = false;
                                        }
                                }
                                break;
                        }

                        if ((coo - q.indexOf("v 玉")) < 11 && (coo - q.indexOf("v 玉")) > 0) {// 玉が
左のとき

                                int dis = ((coo - q.indexOf("v 玉")));//玉までの距離

                                for (int n = 1; n < dis; n++) {
                                        if (!(q.get(coo - (n)).contains("・ "))) {//玉までに空マス
が無かったとき false を返す

                                                boo = false;
                                        }
                                }
                                break;
                        }

                        if ((coo - q.indexOf("v 玉")) > -11 && (coo - q.indexOf("v 玉")) < 0) {// 玉が
右のとき

                                int dis = (-(coo - q.indexOf("v 玉")));//玉までの距離

                                for (int n = 1; n < dis; n++) {
                                        if (!(q.get(coo + (n)).contains("・ "))) {//玉までに空マス
が無かったとき false を返す

                                                boo = false;
                                        }
                                }
                                break;
                        }

                        if ((coo - q.indexOf("v 玉")) < -11) {// 玉が下のとき

                                int dis = (-(coo - q.indexOf("v 玉")) / 11);//玉までの距離

                                for (int n = 1; n < dis; n++) {
                                        if (!(q.get(coo + (11 * n)).contains("・ "))) {//玉までに空
マスが無かったとき false を返す

                                                boo = false;
                                        }
                                }
                                break;
                        }

                }
```

```
		}

		return boo;

	}

	boolean checkB(ArrayList<String> q, int coo) { //角

		boolean boo = false;

		for (int c = 0; c < bishop.length; c++) {
			if ((coo + bishop[c]) == q.indexOf("v 玉")) {
				boo = true;

				if (((coo - q.indexOf("v 玉")) % 12 == 0) && (coo - q.indexOf("v 玉")) > 0)
{// 玉が左上

					int dis = ((coo - q.indexOf("v 玉")) / 12);//玉までの距離

					for (int n = 1; n < dis; n++) {
						if (!(q.get(coo - (12 * n)).contains("・ "))) {//玉までに空
マスが無かったとき false を返す

							boo = false;
						}
					}
					break;
				}

				if (((coo - q.indexOf("v 玉")) % 10 == 0) && (coo - q.indexOf("v 玉")) > 0)
{// 玉が右上

					int dis = ((coo - q.indexOf("v 玉")) / 10);//玉までの距離

					for (int n = 1; n < dis; n++) {
						if (!(q.get(coo - (10 * n)).contains("・ "))) {//玉までに空
マスが無かったとき false を返す

							boo = false;
						}
					}
					break;
				}

				if (((coo - q.indexOf("v 玉")) % 10 == 0) && (coo - q.indexOf("v 玉")) < 0)
{// 玉が左下

					int dis = (-(coo - q.indexOf("v 玉")) / 10);//玉までの距離

					for (int n = 1; n < dis; n++) {
						if (!(q.get(coo + (10 * n)).contains("・ "))) {//玉までに空
マスが無かったとき false を返す

							boo = false;
```

```
                                        }
                                    }
                                    break;
                                }

                                if (((coo - q.indexOf("v 玉")) % 12 == 0) && (coo - q.indexOf("v 玉")) < 0)
{// 玉が右下

                                    int dis = (-(coo - q.indexOf("v 玉")) / 12);//玉までの距離

                                    for (int n = 1; n < dis; n++) {
                                        if (!(q.get(coo + (12 * n)).contains("・ "))) {//玉までに空
マスが無かったとき false を返す

                                            boo = false;
                                        }
                                    }
                                    break;
                                }
                            }
                        }

                        return boo;

                    }

                    boolean checkTP(ArrayList<String> q, int coo) {//と金
                        boolean boo = false;

                        for (int c = 0; c < turn_pawn.length; c++) {
                            if ((coo + turn_pawn[c]) == q.indexOf("v 玉")) {
                                boo = true;
                                break;
                            }
                        }

                        return boo;

                    }

                    boolean checkTL(ArrayList<String> q, int coo) {//成り香
                        boolean boo = false;

                        for (int c = 0; c < turn_lance.length; c++) {
                            if ((coo + gold[c]) == q.indexOf("v 玉")) {
                                boo = true;
                                break;
                            }
                        }

                        return boo;

                    }

                    boolean checkTN(ArrayList<String> q, int coo) {//成り圭
                        boolean boo = false;
```

```
		for (int c = 0; c < turn_knight.length; c++) {
			if ((coo + turn_knight[c]) == q.indexOf("v 玉")) {
				boo = true;
				break;
			}
		}

		return boo;

	}

	boolean checkTS(ArrayList<String> q, int coo) {//成り銀
		boolean boo = false;

		for (int c = 0; c < turn_silver.length; c++) {
			if ((coo + turn_silver[c]) == q.indexOf("v 玉")) {
				boo = true;
				break;
			}
		}

		return boo;
	}

	boolean checkTR(ArrayList<String> q, int coo) {//龍
		boolean boo = false;

		for (int c = 0; c < turn_rook.length; c++) {
			if ((coo + turn_rook[c]) == q.indexOf("v 玉")) {
				if ((coo - 12 == q.indexOf("v 玉")) || (coo - 10 == q.indexOf("v 玉"))
						|| (coo + 12 == q.indexOf("v 玉")) || (coo + 10 ==
q.indexOf("v 玉"))) { //斜めの効きに王がいると true を返すき
					boo = true;
					break;
				}
				boo = true;

				if ((coo - q.indexOf("v 玉")) > 10) {// 玉が上のとき

					int dis = ((coo - q.indexOf("v 玉")) / 11);//玉までの距離

					for (int n = 1; n < dis; n++) {
						if (!(q.get(coo - (11 * n)).contains("・ "))) {//玉までに空
マスが無かったとき false を返す

							boo = false;
						}
					}
					break;
				}

				if ((coo - q.indexOf("v 玉")) < 10 && (coo - q.indexOf("v 玉")) > 0) {// 玉が
左のとき

					int dis = ((coo - q.indexOf("v 玉")));//玉までの距離
```

```
                                        for (int n = 1; n < dis; n++) {
                                                if (!(q.get(coo - (n)).contains("・ "))) {//玉までに空マス
が無かったとき false を返す

                                                        boo = false;
                                                }
                                        }
                                        break;
                                }

                                if ((coo - q.indexOf("v 玉")) > -10 && (coo - q.indexOf("v 玉")) < 0) {// 玉が
右のとき

                                        int dis = (-(coo - q.indexOf("v 玉")));//玉までの距離

                                        for (int n = 1; n < dis; n++) {
                                                if (!(q.get(coo + (n)).contains("・ "))) {//玉までに空マス
が無かったとき false を返す

                                                        boo = false;
                                                }
                                        }
                                        break;
                                }

                                if ((coo - q.indexOf("v 玉")) < -10) {// 玉が下のとき

                                        int dis = (-(coo - q.indexOf("v 玉")) / 11);//玉までの距離

                                        for (int n = 1; n < dis; n++) {
                                                if (!(q.get(coo + (11 * n)).contains("・ "))) {//玉までに空
マスが無かったとき false を返す

                                                        boo = false;
                                                }
                                        }
                                        break;
                                }

                                break;
                        }
                }

                return boo;

        }

        boolean checkTB(ArrayList<String> q, int coo) {//馬
                boolean boo = false;

                for (int c = 0; c < turn_bishop.length; c++) {
                        if ((coo + turn_bishop[c]) == q.indexOf("v 玉")) {
                                boo = true;
```

```
                        if ((coo - 11 == q.indexOf("v 玉")) || (coo - 1 == q.indexOf("v 玉"))
                                || (coo + 1 == q.indexOf("v 玉")) || (coo + 11 ==
q.indexOf("v 玉"))) {//上下左右の効きに王がいると true を返すき
                                boo = true;
                                break;
                        }

                        if (((coo - q.indexOf("v 玉")) % 12 == 0) && (coo - q.indexOf("v 玉")) > 0)
{// 玉が左上

                                int dis = ((coo - q.indexOf("v 玉")) / 12);//玉までの距離

                                for (int n = 1; n < dis; n++) {
                                        if (!(q.get(coo - (12 * n)).contains(" ・ "))) {//玉までに空
マスが無かったとき false を返す

                                                boo = false;
                                        }
                                }
                                break;
                        }

                        if (((coo - q.indexOf("v 玉")) % 10 == 0) && (coo - q.indexOf("v 玉")) > 0)
{// 玉が右上

                                int dis = ((coo - q.indexOf("v 玉")) / 10);//玉までの距離

                                for (int n = 1; n < dis; n++) {
                                        if (!(q.get(coo - (10 * n)).contains(" ・ "))) {//玉までに空
マスが無かったとき false を返す

                                                boo = false;
                                        }
                                }
                                break;
                        }

                        if (((coo - q.indexOf("v 玉")) % 10 == 0) && (coo - q.indexOf("v 玉")) < 0)
{// 玉が左下

                                int dis = (-(coo - q.indexOf("v 玉")) / 10);//玉までの距離

                                for (int n = 1; n < dis; n++) {
                                        if (!(q.get(coo + (10 * n)).contains(" ・ "))) {//玉までに空
マスが無かったとき false を返す

                                                boo = false;
                                        }
                                }
                                break;
                        }

                        if (((coo - q.indexOf("v 玉")) % 12 == 0) && (coo - q.indexOf("v 玉")) < 0)
{// 玉が右下
```

```
                                        int dis = (-(coo - q.indexOf("v 玉")) / 12);//玉までの距離

                                        for (int n = 1; n < dis; n++) {
                                                if (!(q.get(coo + (12 * n)).contains("・ "))) {//玉までに空
マスが無かったとき false を返す

                                                        boo = false;
                                                }
                                        }
                                        break;
                                }
                                break;
                        }
                }

                return boo;

        }

        /*
         *
         * moveMyKoma メソッド
         * 与えられた局面から先手が王手となる手を指す
         * @ return 王手をかけた局面を全て含めた配列
         *
         */
        ArrayList<kyokumen> moveMyKoma(ArrayList<String> q, int coo, ArrayList<String> motiS,
ArrayList<String> motiG) {

                ArrayList<String> test = new ArrayList<String>();

                for (int n = 0; n < q.size(); n++) {
                        test.add(q.get(n));
                }

                ArrayList<String> mo = new ArrayList<String>();

                for (int n = 0; n < motiS.size(); n++) {
                        mo.add(motiS.get(n));

                }

                ArrayList<kyokumen> sama = new ArrayList<kyokumen>();

                Solve s = new Solve();

                int thisKoma[];

                if (coo == -1) { //座標が-1 のときは持ち駒を使用した王手
                        int loop = mo.size();
                        for (int w = 0; w < loop; w++) {

                                for (int m = 0; m < test.size(); m++) {
                                        if ((test.get(m).contains("・ "))) {
                                                test.set(m, mo.get(w));
```

```
                                        mo.remove(0);

                                        if (s.checkOte(test)) {

                                                ArrayList<String> jj = new
ArrayList<String>();
                                                for (int n = 0; n < test.size(); n++) {
                                                        jj.add(test.get(n));
                                                }
                                                ArrayList<String> mm = new
ArrayList<String>();
                                                for (int n = 0; n < mo.size(); n++) {
                                                        mm.add(mo.get(n));
                                                }
                                                kyokumen teq = new kyokumen();
                                                teq.qq = jj;
                                                teq.motiS = mm;
                                                teq.motiG = motiG;
                                                sama.add(teq);

                                                mo.add(test.get(m));
                                                test.set(m, "・ ");

                                        }
                                        if (!s.checkOte(test)) {

                                                mo.add(test.get(m));
                                                test.set(m, "・ ");
                                        }

                                }
                        }
                }
        } else {

                thisKoma = komaK[myKomaList.indexOf(test.get(coo))];// 駒の種類

                for (int m = 0; m < thisKoma.length; m++) {

                        if (test.get((coo + thisKoma[m])).contains("x 外") || test.get((coo +
thisKoma[m])).contains("^")) {// 可動範囲が盤外か自駒があるとき

                                //離れ駒は一方向探索したとき、それ以上進めないならば、
次の方向の探索を行う
                                if ((q.get(coo)).equals(("^香"))) {
                                        break;
                                } else if ((q.get(coo)).equals(("^飛")) || (q.get(coo)).equals(("^角")))
{
                                        if (m <= 8) {
                                                m = 8;
                                        }
                                        if (m >= 9 && m <= 17) {
                                                m = 17;
                                        }
                                        if (m >= 18 && m <= 26) {
```

```
                                        m = 26;
                                }
                                if (m > 27) {
                                        break;
                                }
                        } else if ((q.get(coo)).equals(("^龍")) || (q.get(coo)).equals(("^馬")))
{
                                if (m <= 8) {
                                        m = 8;
                                }
                                if (m >= 9 && m <= 17) {
                                        m = 17;
                                }
                                if (m >= 18 && m <= 26) {
                                        m = 26;
                                }
                                if (m >= 27 && m <= 35) {
                                        m = 35;
                                }

                        }

                } else if (test.get((coo + thisKoma[m])).contains("v")) {// 敵駒があるとき

                        // その駒を自分の手持ちにセット
                        mo.add((myKomaList.get(enemyKomaList.indexOf(test.get(coo +
thisKoma[m])))));
                        // その駒の行き先にセット
                        test.set((coo + thisKoma[m]), test.get(coo));
                        // その駒のあったところを空に
                        test.set(coo, "・ ");

                        if (!(s.checkOte(test))) {
                                test.set(coo, test.get(coo + thisKoma[m]));
                                test.set((coo + thisKoma[m]),
enemyKomaList.get(myKomaList.indexOf(mo.get(mo.size() - 1))));
                                mo.remove(mo.size() - 1);

                                //離れ駒は一方向探索したとき、それ以上進めない
ならば、次の方向の探索を行う
                                if ((q.get(coo)).equals(("^香"))) {
                                        break;
                                } else if ((q.get(coo)).equals(("^飛")) ||
(q.get(coo)).equals(("^角"))) {
                                        if (m <= 8) {
                                                m = 8;
                                        }
                                        if (m >= 9 && m <= 17) {
                                                m = 17;
                                        }
                                        if (m >= 18 && m <= 26) {
                                                m = 26;
                                        }
                                        if (m > 27) {
                                                break;
```

```
                                        }
                                } else if ((q.get(coo)).equals(("^龍")) ||
(q.get(coo)).equals(("^馬"))) {
                                        if (m <= 8) {
                                                m = 8;
                                        }
                                        if (m >= 9 && m <= 17) {
                                                m = 17;
                                        }
                                        if (m >= 18 && m <= 26) {
                                                m = 26;
                                        }
                                        if (m >= 27 && m <= 35) {
                                                m = 35;
                                        }

                                }

                        } else if (s.checkOte(test)) {

                                ArrayList<String> jj = new ArrayList<String>();
                                for (int n = 0; n < test.size(); n++) {
                                        jj.add(test.get(n));
                                }
                                ArrayList<String> mm = new ArrayList<String>();
                                for (int n = 0; n < mo.size(); n++) {
                                        mm.add(mo.get(n));
                                }
                                kyokumen teq = new kyokumen();
                                teq.qq = jj;
                                teq.motiS = mm;
                                teq.motiG = motiG;
                                sama.add(teq);

                                test.set(coo, test.get(coo + thisKoma[m]));
                                test.set((coo + thisKoma[m]),
enemyKomaList.get(myKomaList.indexOf(mo.get(mo.size() - 1))));
                                // break;
                                mo.remove(mo.size() - 1);

                                //離れ駒は一方向探索したとき、それ以上進めない
ならば、次の方向の探索を行う
                                if ((q.get(coo)).equals(("^香"))) {
                                        break;
                                } else if ((q.get(coo)).equals(("^飛")) ||
(q.get(coo)).equals(("^角"))) {
                                        if (m <= 8) {
                                                m = 8;
                                        }
                                        if (m >= 9 && m <= 17) {
                                                m = 17;
                                        }
                                        if (m >= 18 && m <= 26) {
                                                m = 26;
                                        }
                                        if (m > 27) {
```

```
                                                        break;
                                                }
                                        } else if ((q.get(coo)).equals(("^龍")) ||
(q.get(coo)).equals(("^馬"))) {
                                                if (m <= 8) {
                                                        m = 8;
                                                }
                                                if (m >= 9 && m <= 17) {
                                                        m = 17;
                                                }
                                                if (m >= 18 && m <= 26) {
                                                        m = 26;
                                                }
                                                if (m >= 27 && m <= 35) {
                                                        m = 35;
                                                }

                                        }

                                }

                        } else if (test.get((coo + thisKoma[m])).contains("・ ")) {
                                // その駒の行き先にセット
                                test.set((coo + thisKoma[m]), test.get(coo));
                                // その駒のあったところを空に
                                test.set(coo, "・ ");

                                if (!(s.checkOte(test))) {
                                        test.set(coo, test.get(coo + thisKoma[m]));
                                        test.set((coo + thisKoma[m]), "・ ");

                                } else if (s.checkOte(test)) {

                                        ArrayList<String> ll = new ArrayList<String>();
                                        for (int n = 0; n < test.size(); n++) {
                                                ll.add(test.get(n));
                                        }
                                        kyokumen teq = new kyokumen();
                                        ArrayList<String> mm = new ArrayList<String>();
                                        for (int n = 0; n < mo.size(); n++) {
                                                mm.add(mo.get(n));
                                        }
                                        teq.qq = ll;
                                        teq.motiS = mm;
                                        teq.motiG = motiG;
                                        sama.add(teq);

                                        test.set(coo, test.get(coo + thisKoma[m]));
                                        test.set((coo + thisKoma[m]), "・ ");

                                }
                        }

                }

        }
```

```
		return sama;

	}

	/*
	 *
	 * moveEnemyKoma メソッド
	 * 与えられた局面から後手が王手を回避する手を指す
	 * @ return 王手を回避した局面を全て含めた配列
	 *
	 */
	ArrayList<kyokumen> moveEnemyKoma(ArrayList<String> q, int coo, ArrayList<String> motiS,
ArrayList<String> motiG) {

		ArrayList<String> test = new ArrayList<String>();
		for (int n = 0; n < q.size(); n++) {
			test.add(q.get(n));
		}

		ArrayList<String> mo = new ArrayList<String>();

		for (int n = 0; n < motiG.size(); n++) {
			mo.add(motiG.get(n));
		}

		ArrayList<kyokumen> sama = new ArrayList<kyokumen>();

		Solve s = new Solve();

		int thisKoma[];

		if (coo == -1) {

			int loop = mo.size();
			for (int w = 0; w < loop; w++) {

				for (int m = 0; m < test.size(); m++) {
					if ((test.get(m).contains("・"))) {
						test.set(m, mo.get(w));
						mo.remove(0);

						if (!s.checkOte(test)) {

							ArrayList<String> jj = new
ArrayList<String>();

							for (int n = 0; n < test.size(); n++) {
								jj.add(test.get(n));
							}
							kyokumen teq = new kyokumen();
							ArrayList<String> mm = new
ArrayList<String>();

							for (int n = 0; n < mo.size(); n++) {
								mm.add(mo.get(n));
							}
							teq.qq = jj;
							teq.motiS = motiS;
```

```
                                                teq.motiG = mm;
                                                sama.add(teq);

                                                mo.add(test.get(m));
                                                test.set(m, "・ ");

                                        }
                                        if (s.checkOte(test)) {
                                                mo.add(test.get(m));
                                                test.set(m, "・ ");
                                        }

                                }
                        }
                }

        } else {

                thisKoma = komaK[enemyKomaList.indexOf(test.get(coo))];// その駒の種類

                for (int m = 0; m < thisKoma.length; m++) {
                        if (test.get((coo + -thisKoma[m])).contains("x 外") || test.get((coo +
-thisKoma[m])).contains("v")) {// 動く先が盤外か自駒があるとき

                                if ((q.get(coo)).equals(("v 香"))) {
                                        break;
                                } else if ((q.get(coo)).equals(("v 飛")) || (q.get(coo)).equals(("v 角
"))) {
                                        if (m < 8) {
                                                m = 8;
                                        }
                                        if (m > 9 && m < 17) {
                                                m = 17;
                                        }
                                        if (m > 18 && m < 26) {
                                                m = 26;
                                        }
                                        if (m > 26) {
                                                break;
                                        }
                                } else if ((q.get(coo)).equals(("v 龍")) || (q.get(coo)).equals(("v 馬
"))) {
                                        if (m < 8) {
                                                m = 8;
                                        }
                                        if (m > 9 && m < 17) {
                                                m = 17;
                                        }
                                        if (m > 18 && m < 26) {
                                                m = 26;
                                        }
                                        if (m > 26 && m < 35) {
                                                m = 35;
                                        }

                                }
```

```
} else if (test.get((coo + -thisKoma[m])).contains("^")) {// 敵駒があるとき

// その駒を自分の手持ちにセット
mo.add((enemyKomaList.get(myKomaList.indexOf(test.get(coo +
-thisKoma[m])))));
// その駒の行き先にセット
test.set((coo + -thisKoma[m]), test.get(coo));
// その駒のあったところを空に
test.set(coo, "・ ");

if (s.checkOte(test)) {

test.set(coo, test.get(coo + -thisKoma[m]));
test.set((coo + -thisKoma[m]),
myKomaList.get(enemyKomaList.indexOf(mo.get(mo.size() - 1))));
mo.remove(mo.size() - 1);

} else if (!(s.checkOte(test))) {
ArrayList<String> ll = new ArrayList<String>();
for (int n = 0; n < test.size(); n++) {
ll.add(test.get(n));
}
kyokumen teq = new kyokumen();
ArrayList<String> mm = new ArrayList<String>();
for (int n = 0; n < mo.size(); n++) {
mm.add(mo.get(n));
}
teq.qq = ll;
teq.motiS = motiS;
teq.motiG = mm;
sama.add(teq);

test.set(coo, test.get(coo + -thisKoma[m]));
test.set((coo + -thisKoma[m]),
myKomaList.get(enemyKomaList.indexOf(mo.get(mo.size() - 1))));
mo.remove(mo.size() - 1);

}

} else if (test.get((coo + -thisKoma[m])).contains("・ ")) {

// その駒の行き先にセット
test.set((coo + -thisKoma[m]), test.get(coo));
// その駒のあったところを空に
test.set(coo, "・ ");

if (s.checkOte(test)) {

test.set(coo, test.get(coo + -thisKoma[m]));
test.set((coo + -thisKoma[m]), "・ ");

} else if (!(s.checkOte(test))) {

ArrayList<String> ll = new ArrayList<String>();
```

```
                                                for (int n = 0; n < test.size(); n++) {
                                                        ll.add(test.get(n));
                                                }
                                                kyokumen teq = new kyokumen();
                                                ArrayList<String> mm = new ArrayList<String>();
                                                for (int n = 0; n < mo.size(); n++) {
                                                        mm.add(mo.get(n));
                                                }
                                                teq.qq = ll;
                                                teq.motiS = motiS;
                                                teq.motiG = mm;
                                                sama.add(teq);

                                                test.set(coo, test.get(coo + -thisKoma[m]));
                                                test.set((coo + -thisKoma[m]), "・ ");
                                        }

                                }
                        }
                }

                return sama;

        }
}
```

「Solve クラス」

```
   package sotuken;

import java.util.ArrayDeque;
import java.util.ArrayList;
import java.util.Deque;


/*
 * Solve クラス
 * 与えられた局面から詰みを探索する
 *
 */
public class Solve {

        Deque<kyokumen> stack = new ArrayDeque<kyokumen>();
        Deque<Integer> limitList = new ArrayDeque<Integer>();

        Koma koma = new Koma();
/*
 * checkOte メソッド
 * 現在の盤面で王手がかかっているかを判定するメソッド
 *
 */
```

```
boolean checkOte(ArrayList<String> q) {

		boolean ote = false;

		for (int n = 0; n < q.size(); n++) {

			if (q.get(n).contains("^")) {
				if (q.get(n).equals("^歩")) {
					if (koma.checkP(q, n)) {

						ote = true;

						break;
					}
					;
				}
				if (q.get(n).equals("^香")) {
					if (koma.checkL(q, n)) {
						ote = true;
						break;
					}
					;
				}
				if (q.get(n).equals("^桂")) {
					if (koma.checkN(q, n)) {
						ote = true;
						break;
					}
					;
				}
				if (q.get(n).equals("^銀")) {
					if (koma.checkS(q, n)) {
						ote = true;
						break;
					}
					;
				}
				if (q.get(n).equals("^金")) {
					if (koma.checkG(q, n)) {
						ote = true;
						break;
					}
					;
				}
				if (q.get(n).equals("^玉")) {
					if (koma.checkK(q, n)) {
						ote = true;
						break;
					}
					;
				}
				if (q.get(n).equals("^飛")) {
					if (koma.checkR(q, n)) {
						ote = true;
						break;
					}
```

```
                        ;
                }
                if (q.get(n).equals("^角")) {
                        if (koma.checkB(q, n)) {
                                ote = true;
                                break;
                        }
                        ;
                }
                if (q.get(n).equals("^と")) {
                        if (koma.checkTP(q, n)) {
                                ote = true;
                                break;
                        }
                        ;
                }
                if (q.get(n).equals("^杏")) {
                        if (koma.checkTL(q, n)) {
                                ote = true;
                                break;
                        }
                        ;
                }
                if (q.get(n).equals("^圭")) {
                        if (koma.checkTN(q, n)) {
                                ote = true;
                                break;
                        }
                        ;
                }
                if (q.get(n).equals("^全")) {
                        if (koma.checkTS(q, n)) {
                                ote = true;
                                break;
                        }
                        ;
                }
                if (q.get(n).equals("^龍")) {
                        if (koma.checkTR(q, n)) {
                                ote = true;
                                break;
                        }
                        ;
                }
                if (q.get(n).equals("^馬")) {
                        if (koma.checkTB(q, n)) {
                                ote = true;
                                break;
                        }
                        ;
                }

        }

}
```

```
		return ote;

	}

	boolean notOte = false;

	/*
	 * moveSente メソッド
	 * 先手の駒を動かす
	 */
	ArrayList<kyokumen> moveSente(ArrayList<String> q, ArrayList<String> motiS,
ArrayList<String> motiG) {

		ArrayList<ArrayList<kyokumen>> ansListList = new
ArrayList<ArrayList<kyokumen>>();
		ArrayList<kyokumen> ansList = new ArrayList<kyokumen>();
		ArrayList<String> answer = new ArrayList<String>();
		for (int n = 0; n < q.size(); n++) {
			answer.add(q.get(n));
		}

		ArrayList<Integer> myk = new ArrayList<Integer>();

		for (int n = 0; n < q.size(); n++) {

			if (q.get(n).contains("^")) {//盤上の先手駒の位置を把握

				myk.add(n);

			}
		}
		myk.add(-1); //持ち駒の使用
		for (int n = 0; n < myk.size(); n++) {//王手となるてを全て ansListList に収容

			ansListList.add(koma.moveMyKoma(answer, myk.get(n), motiS, motiG));

		}

		for (int n = 0; n < ansListList.size(); n++) {
			ansList.addAll(ansListList.get(n));
		}

		return ansList;
	}

	/*
	 * moveGote メソッド
	 * 後手の駒を動かす
	 */
	ArrayList<kyokumen> moveGote(ArrayList<String> q, ArrayList<String> motiS,
ArrayList<String> motiG) {

		ArrayList<ArrayList<kyokumen>> ansListList = new
ArrayList<ArrayList<kyokumen>>();
		ArrayList<kyokumen> ansList = new ArrayList<kyokumen>();
		ArrayList<String> answer = new ArrayList<String>();
```

```
		for (int n = 0; n < q.size(); n++) {
			answer.add(q.get(n));
		}

		ArrayList<Integer> myk = new ArrayList<Integer>();

		for (int n = 0; n < q.size(); n++) {

			if (q.get(n).contains("v")) {//盤上の先手駒の位置を把握
				myk.add(n);

			}
		}
		myk.add(-1); //持ち駒の使用
		for (int n = 0; n < myk.size(); n++) {//王手を回避する手を全て ansListList に収容

			ansListList.add(koma.moveEnemyKoma(answer, myk.get(n), motiS,
motiG));

		}
		for (int n = 0; n < ansListList.size(); n++) {
			ansList.addAll(ansListList.get(n));
		}

		return ansList;
	}

	boolean bb = false;

	/*
	 * getMove メソッド
	 * 局面から一手進める
	 * @ return 局面
	 */
	ArrayList<kyokumen> getMove(ArrayList<String> q, ArrayList<String> motiS,
ArrayList<String> motiG) {

		ArrayList<kyokumen> ans = new ArrayList<kyokumen>();

		if (limit % 2 == 0) {//深さが偶数のとき、先手が指す

			ans = moveSente(q, motiS, motiG);

			if (ans.isEmpty()) {
				notOte = true;


			}
		} else if (limit % 2 == 1) {//深さが奇数のとき、後手が指す

			ans = moveGote(q, motiS, motiG);

			if (ans.isEmpty()) {
				bb = true;
```

```
                }
            }

            return ans;

        }

        int limit = 0;//読み込む深さの制限

        /*
         * solve メソッド
         * 受け取った局面から解を導きだす
         * @return 詰み上がり図をおさめた配列
         */
        ArrayList<kyokumen> solve(ArrayList<String> q, ArrayList<String> motiS, ArrayList<String>
motiG) {

            ArrayList<kyokumen> answerList = new ArrayList<kyokumen>();
            int maxLim = 4;//深さの制限
            kyokumen kore = new kyokumen();
            kore.qq = q;
            kore.motiS = motiS;
            kore.motiG = motiG;

            stack.add(kore);
            limitList.add(0);

            boolean tumi = false;

            while (!stack.isEmpty()) {

                kyokumen node = stack.pop();

                limit = limitList.pop();

                ArrayList<kyokumen> a = getMove(node.qq, node.motiS, node.motiG);
                int lim = limit + 1;//局面の深さ


                if (tumi || bb) {

                    kyokumen answer = new kyokumen();

                    ArrayList<String> jj = new ArrayList<String>();
                    for (int n = 0; n < node.qq.size(); n++) {
                        jj.add(node.qq.get(n));
                    }
                    ArrayList<String> mms = new ArrayList<String>();
                    for (int n = 0; n < node.motiS.size(); n++) {
                        mms.add(node.motiS.get(n));
                    }
                    ArrayList<String> mmg = new ArrayList<String>();
                    for (int n = 0; n < node.motiG.size(); n++) {
                        mmg.add(node.motiG.get(n));
                    }
```

```
                                answer.qq = jj;
                                answer.motiS = mms;
                                answer.motiG = mmg;

                                answerList.add(answer);
                                System.out.println(node.motiG);
                                System.out.println(node.qq);
                                System.out.println(node.motiS);
                                // break;
                                bb = false;
                        }

                        if (limit <= maxLim) {//深さ maxLim まで読む

                                for (int h = 0; h < a.size(); h++) {

                                        stack.push(a.get(h));
                                        limitList.push(lim);

                                        if (bb) {

                                                tumi = true;
                                                break;
                                        }

                                }

                        } else if (limit > maxLim) {

                                limit--;

                        }

                }

                return answerList;

        }

}
```

「kyokumen クラス」

```
package sotuken;

import java.util.ArrayList;

/*
 * kyokumen クラス
 * 盤面、持ち駒を表すクラス
 */
public class kyokumen {
```

```
        public ArrayList<String> qq;//盤面
        public ArrayList<String> motiS;//先手持ち駒
        public ArrayList<String> motiG;//後手持ち駒

}
```