卒業研究報告書

題目

オセロにおけるライバルAIの作成について

指導教員

石水隆　講師

報告者

13-1-037-0101

松村 憲樹

近畿大学理工学部情報学科

平成29年01月31日提出

概要

本研究は，オセロにおいて対戦相手と対戦の中で同じような実力に調整することで，対戦相手にとってライバルになれるような AI，通称ライバル AI を作成し，そのライバル AI の性能を検査する．ライバル AI は対戦相手の実力をチェックするために対戦相手の打った手を評価し，その評価値の高さによって自身の打つ手を変える．

本研究では Java 言語でライバル AI を作成する．本研究で作成するライバル AI は各候補手に対して数手先の局面を生成し，の評価をするために α β 法を用いてその候補手の評価値を求める．ライバル AI は，対戦相手が打った手に対し、その手と同等の評価値を持つ手を打つことで、対戦相手と同等の実力となるようにする．

作成したライバル AI の性能を検査するために，ライバル AI が様々な実力の AI および人間の対戦相手と先手後手 100 戦，10 人の人間と先手後手 10 戦それぞれ対戦し，ライバル AI が対戦相手に合わせて実力を変動させ，勝率が 5 割程度になっているか検査する．

目次

1　序論

1.1 本研究の背景

オセロのAIは日々進化しており，計算機の性能や探索アルゴリズムの発展と向上によって人間のトッププレイヤーにも負けないような強いオセロAIを作成することが可能なレベルに達している．そのため，人間がオセロAIと戦う場合は強さのレベルを下げて戦うということになる．しかし，ここで自分の実力に合わせた適正なレベルに下げないと，強すぎて手も足も出ず全く勝てなかったり，逆に弱すぎてあっさり勝ってしまう可能性がある．

1.2 本研究の目的

1.1節で述べたように人間がAIと相手にオセロを楽しむAIをプレイヤーの実力にあった適正なレベルに設定する必要がある．適正なレベルに調整するためにはオセロAIと人間が数局対戦し，自分のレベルに探す必要があり，適正なレベルのオセロAIと対戦するのに時間がかかってしまう．そこで，本研究では対戦中に対戦相手のレベルに自身のレベルが近くなるように自動調整し，対戦相手と同じ実力になるようなAI，通称ライバル AIを作成し，そのライバルAIの性能を検査する．

1.3 オセロの関する既知の結果

本節では、オセロの関する既知の結果を示す.

1.3.1　盤面の評価方法

オセロのAIは打つ手を決めるために盤面の評価を行う．評価する基準として，自分の石の数，取られないことが確定している確定石の数，マス目に点数を振り分けて石の数と組み合わせた値の合計等様々な評価基準があるが，未だにどのような評価基準を用いれば最善手が打てるようになるかは解決されていない.

1.3.2　先読み

オセロのAIがより良い手を打てるようにするために自分の打つ手を先読みし，先読みした盤面の評価を行いより自分にとって最善手を打つことができる．先読みをする方法として探索アルゴリズムのミニマックス法，$\alpha\beta$ 法と特殊な計算方法のモンテカルロ法がある．$\alpha\beta$ 法については2.2節にて説明する．ミニマックス法は自分が常に評価が最大に，相手が常に最小になるように探索を行う探索アルゴリズムである．モンテカルロ法は乱数を用いたシミュレーションを複数回行い，理想の手に近似していく計算方法がある．主にミニマックス法と $\alpha\beta$ 法が用いられている.

1.3.3　ライバルAI

相手において実力を変動させるライバル AI は参考文献[2]において，対戦相手とのプレイ記録から機械学習を用いて学習し，様々な戦略を打つようにするために遺伝的アルゴリズムを用いて，最適化を行い，学習したAIを再度同じ対戦相手と対戦した結果，全ての人が勝率5割程度にすることはできなかった[2].

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.

まず第2章にて，本研究で作成したライバルAIとライバルAIの性能の検査に用いる3つのAIの仕様について述べ，第3章では作成したライバルAIの性能を検査するために，ライバルAIとライバル

AI 自身を含めた 4 つの AI が対戦した結果を述べる．そして第 5 章では，第 4 章の結果に対する考察と結論を述べる．

2　研究内容

本章では本研究にて作成したオセロの AI プログラムについて述べる．本研究ではオセロのライバル AI(以下 RvlAI とする)の作成のために java 言語を用いた．本研究で作成した RvlAI は対戦相手の打った手の評価値の高さによって自身の打つ手を変える AI である．RvlAI は各候補手に評価値を設定し，対戦相手が評価値の高い手を打てば，RvlAI も自分の候補手から評価値の高い手を打ち，逆に対戦相手が評価値の低い手を打てば，自分も評価値の低い手を打つようにすることで，対戦相手と同程度の実力になるようにする．また，RvlAI は候補手の先読みの方法として α β 法を用いている．候補手の評価方法と α β 法については以下の 2.1 節と 2.2 節で述べる．本研究で作成されたプログラムのソースコードを付録に添付する．

2.1 候補手の評価方法

RvlAI には，対戦相手と自分の実力を同じようにするために，候補手に対して評価値を設定することができる．その評価方法をこの節で述べる．各マス目に点数をふった．マス目の点数にそのマス目に置いてある石が黒ならば 1 を，白ならば-1 を，何も置かれていなければ 0 を掛ける．マス目ごとのあたいの合計を評価値とした．図 1 は各マス目の点数を示している．一般にオセロでは角を取れば有利であり，角を取れるときには積極的に打つべきとされる．一方角から斜めに 1 マス内側のマス(b2, g2, b7, g7)はＸマスと呼ばれ，ここに打つＸ打ちは大変危険な手で避けるべきとされる。そこで本研究では，図 1 のように角に高いプラス点を、Ｘマスに大きなマイナス点を設定し，ＡＩが角を優先する，角の手前を取らないようにする，端の行と列を取らせないようにすることでより多くの石を取ることが期待できるからである．

120	-20	20	5	5	20	-20	120
-20	-40	-5	-5	-5	-5	-40	-20
20	-5	15	3	3	15	-5	20
5	-5	3	3	3	3	-5	5
5	-5	3	3	3	3	-5	5
20	-5	15	3	3	15	-5	20
-20	-40	-5	-5	-5	-5	-40	-20
120	-20	20	5	5	20	-20	120

図 1:各マス目の点数

2.2 α β 法

RvlAI には候補手の評価値を先読みするために α β 法が用いられている． α β 法とは探索アルゴリズムの 1 つで，図のように自分の手番の局面を丸，相手の手番の局面を四角形で表し，最下層に割り当てられた評価値を探索し，自分の手番のときは最も評価値が高い値を選び，相手の手番のときは最も値が小さい値を選び，その中からもっとも評価値が高い局面を見つけるアルゴリズムである．また，α β 法は効率良く評価値を選ぶため 1 つ目の評価値が決まり 2 つ目以降の評価値を探索するとき，最小値を求める場合は 1 つ目の評価値より高い値が見つかったとき，最大値を求める場合は 1 つ目の評価値より小さい値が見つかったときにそれぞれその局面以降の探索を行わないようになっている．こ

れにより全てを探索するよりも早く評価値を求めることができる．

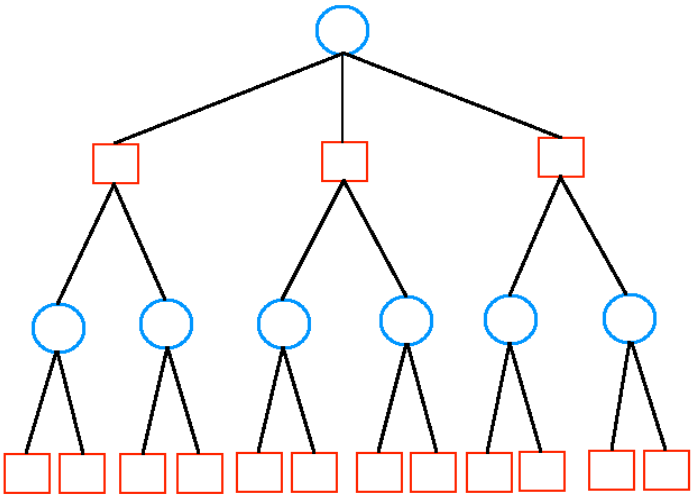

図 2: α β の探索図

3　ライバル AI プログラム

本章では，本研究で作成したライバル AI プログラムについて説明する．

付録に本研究で作成したライバル AI プログラムのソースを示す．

まず，付録にある RvlAI クラスの中にある State クラスと Evaluator クラスについて説明する．次にライバル AI プログラムである RvlAI クラスについて説明する．

3.1 GameState クラス

GameState クラスはオセロの盤面を表すクラスで盤面を表す int 型の配列の data，前の盤面を保存する GameState 型のインスタンスの preBoard，打った手を記録する int 型の配列の log，手番を表す int 型の変数の player，白と黒の石の数をそれぞれ記録する int 型の変数の white と black のフィールドを持つ．指定した場所に石を打つ put メソッド，打った石から自分の石との間に相手の石があるときにその石を自分の石に裏返す reverse メソッド，パスをするか調べる checkPass メソッド，手番を入れ替える pass メソッド，現在の白と黒の石の数を数える countDisc メソッド，現在の盤面をコピーする copy メソッドを持つ．

3.2 Evaluator クラス

Evaluator クラスは AI における評価値を表すクラスで，2.1 節で述べたマス目ごとの点数を設定した int 型の配列の values のフィールドを持つ．values に点数を代入する makeValues メソッドを持つ．

3.3 RvlAI クラス

RvlAI クラスは対戦相手の打った手によって実力を変えるライバル AI を表すクラスである．マス目ごとの評価値を表す Evaluator クラスのインスタンスの eval と自分の手番の石の色を表す int 型の変数の color のフィールドを持つ．コンストラクタで color に黒ならば 1 白ならば-1 を引数に代入し，eval の values に評価値を設定する．

3.3.1alphabeta メソッド

alphabeta メソッドでは 2.2 節で述べた αβ 法を用いて候補手の評価値を返すメソッドである．探索する深さを表す引数 depth が 0 のときに evaluate メソッドを呼び出し，2.1 節で述べたマス目の点数と黒ならば 1，白ならば-1 を乗算し，その値の合計値を盤面の値として返す．パスをする場合は盤面をコピーする GameState クラスの copy メソッドを用いて盤面を引数の state にコピーし，pass メソッドで盤面を相手に変え，depth を 1 減らした値で再帰呼び出しを行い，その値を返す．それ以外は movecheck メソッド候補手を探し，盤面を引数の state にコピーし，value に候補手を打った後の盤面の評価値を value に代入し，value が引数の alpha より大きい場合に alpha に value の値を代入しその値を返す．

3.3.2decide メソッド

decide メソッドは自分の候補手と相手の候補手を照らし合わせ，相手の候補手の強さによって手を変え，どの場所に手を打つか決めるメソッドである．movecheck メソッドを用いて候補手を探し，候補手が 1 つの場合は打つ手の場所の値を返し，パスをしなければならない場合はパスを行う．残りターン数が 6 ターンになると評価基準を自分の石の数と相手の石の数の差がより大きくなるような手を打つようにする．それ以外の場合は alphabeta メソッドと同じように自分と相手の候補手の評価をし，候補手の場所と評価値を ArrayList に格納する．そこで相手が打った手の強さによって自分の打った手と同じような強さを持つ候補手を選び，その値に該当する場所の値を返す．

4　実験結果と考察

本研究では，RvlAI の性能を検証するために，RvlAI 自身を含んだ 4 つの AI と先手後手 100 戦，10 人の人間と先手後手 10 戦ずつ対戦してもらった．本研究の対戦の目的として RvlAI が全ての対戦相手に対して勝率 5 割程度とする．勝率を 5 割程度と設定したのは，RvlAI が対戦相手のレベルに合わせて打つ手を変える AI であることからである．対戦する AI は RvlAI 自身，ランダムに手を打つ AI(以下 RndAI とする)，常に最善手を打つ AI(以下強 AI とする)，常に最悪手を打つ AI(以下弱 AI とする)である．また強 AI と弱 AI は先読みと候補手の評価のために 2.1 節で述べた評価方法と 2.2 節で述べた α β 法を用いている．対戦結果を以下の表 1 と図 3 に示す．表 1 は先手後手 100 戦ずつ対戦した RvlAI のそれぞれの AI に対する勝利数を示す．図 3 は 10 人の人間に RvlAI と先手後手 10 戦ずつ対戦し，勝利数ごとの人数を示す．

表 1 の対戦結果から，先手後手に関わらず，それぞれ一定の勝率を得たことから先手後手の有利はなかったと考えられる．RvlAI が RndAI に対して勝率が高いのは，RndAI がランダムに手を打ってしまうので打つ手のレベルに一貫性が無いからと考えられる．RndAI は RvlAI からすれば，打つ手の評価がよく変わってしまう相手であることから，相手の実力をはっきり把握できずに RvlAI がそのまま勝ってしまうことが多くなってしまうということが考えられる．それに対し，RvlAI，強 AI，弱 AI は先読みと候補手の評価を行うことで，打つ手に一貫性があることで，RvlAI は実力を把握しやすいため，勝率が 5 割前後になったと考えられる．

図 3 の対戦結果から，先手の場合は 2 勝と 3 勝している人が多いことがわかる．しかし，勝率が 5 割程度なのは 10 人中 2 人しかいない．対して後手の場合は 5 勝，6 勝している人が 5 人と半数に達しているが，全員が勝率 5 割程度にはならなかったことがわかる．これらのことから，人によってオセロの盤面の評価基準が異なることから，全員が勝率 5 割にならなかったと考えられる．

表 1:AI 同士の対戦結果(試行回数 100 回)

RvlAI	RndAI	RvlAI	強 AI	弱 AI
先手	74	58	62	46
後手	80	43	44	46

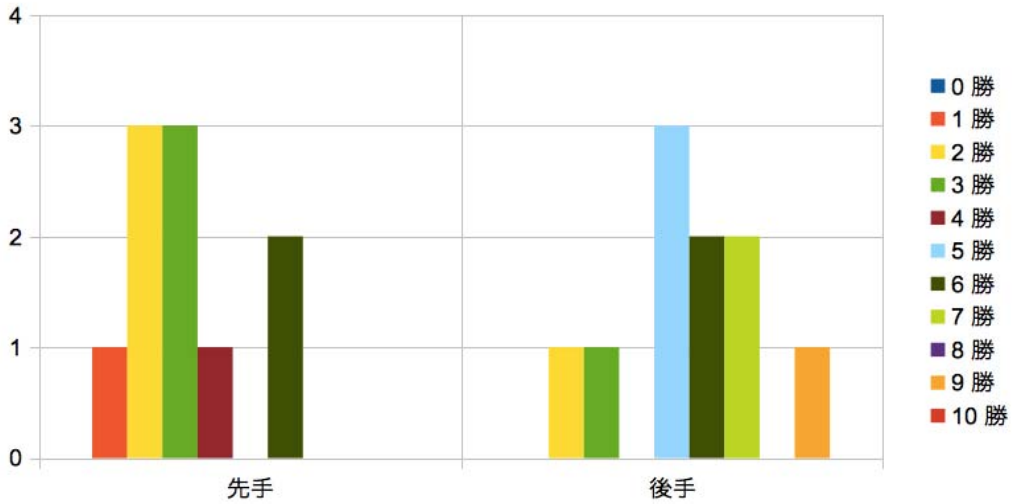

図 3:RvlAI と人との対戦結果(試行回数 10 回)

5 結論

本研究では対戦相手と同じ実力で戦うオセロのライバル AI の作成をした．以上の結果から本研究で製作した RvlAI は全ての対戦相手に対して勝率 5 割程度を達成することができなかった．よって本研究で作成された RvlAI には改善の余地があると考えられる．改善点として，多彩な戦略を持つようにすることがあげられる．対戦結果を確認したところ，複数回対戦していると，戦略が少なく，動きが一辺倒になっているという課題があった．この課題に対してより多い戦略を持たせるために，遺伝的アルゴリズムを用いることで，様々な局面を学習することでより多彩な手を打つようにすることができる．また，定石データベースを取り込むことがあげられる．そうすることで，より多彩な戦略を持つことができるようになると考えられる．他にも前半と後半で評価基準を変えることで，より戦略が広がると考えられる．

参考文献

[1] Seal Software：リバーシのアルゴリズム，工学社(2003)

[2] 上田陽平，池田心：遺伝的アルゴリズムによる人間のレベルに適応する多様なオセロ AI の生成，研究報告 ゲーム情報学，Vol.2012-GI-27，No.5，pp.1-8，情報処理学会 (2012) http://id.nii.ac.jp/1001/00080933/

[3] 大筆豊，オセロプログラムの評価関数の改善について，学会研究報告 ゲーム情報学，Vol.2004-GI-11, No.4, pp.15-20，情報処理学会 (2004) http://id.nii.ac.jp/1001/00058554/

[4] 久保田悠司，佐藤佳州，高橋大介，マルチコアプロセッサと SIMD 演算によるモンテカルロ木探索を用いたオセロの実装，研究報告 ゲーム情報学, Vol.2009-GI-22, No.7, pp.1-8，情報処理学会 (2009) http://id.nii.ac.jp/1001/00062419/

[5] 森田悠樹，橋本剛，小林康幸，オセロ求解に向けた単純な縦型探索をベースにする探索方法の研究，ゲームプログラミングワークショップ 2010 論文集, Vol.2010, No.12, pp.36-41，情報処理学会 (2010) http://id.nii.ac.jp/1001/00071311/

[6] 今田智大，橋本剛，オセロのハンディキャップに関する研究，ゲームプログラミングワークショップ 2012 論文集, Vol.2012, No. 6, pp.151-154 情報処理学会 (2012) http://id.nii.ac.jp/1001/00091345/

付録

本研究で作成したライバル AI のプログラムのソースコードを以下に掲載する.

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.Random;

/**
 * RvlAIクラス
 * @author kazuki
 * 相手の打つ手によって自分の打つ手を変えるライバルAIのクラス
 */
public class RvlAI {
	Evaluator eval = new Evaluator(); // 各マス目の評価値のインスタンス
	int color; //自分の石の色を表す変数

	/**
	 * コンストラクタ
	 * @param color 自分の石の色を代入するための引数
	 */
	public RvlAI(int color) {
		// 評価値の表の生成とコピーする盤面の初期化を行う
		this.color = color;
		eval.makeValues();

	}

	/**
	 * decideメソッド
	 * @param state 現在の盤面をコピーするための引数
	 * @return 打つ手の場所の座標をかえす
	 */
	public int[] decide(GameState state){
		ArrayList<int[]> list = new ArrayList<int[]>();
		Random rnd = new Random();
		int masu[] = new int[2];
		int maxvalue = Integer.MIN_VALUE;
		int minvalue = Integer.MAX_VALUE;
		int eval = 0;
		int counter = 0;
		int range[] = {-1, 0, 1};
		GameState subBoard = new GameState();
		GameState subPreBoard = new GameState();
		ArrayList<Value> v1 = new ArrayList<Value>();
		ArrayList<Value> v2 = new ArrayList<Value>();
```

```
//候補手を探す
movecheck(state, list);

if (list.size() == 1) {
	//候補手が１つの場合にその手を選択する
	masu[0] = list.get(0)[0];
	masu[1] = list.get(0)[1];
	return masu;
}
if((60-state.getTurn()) < 6){
	//残りのターン数が６ターン以下の場合
	for(int i=0; i<list.size(); i++){
		//自分と相手の石の差が大きくなるような手を選ぶ
		subBoard = state.copy();
		subBoard.put(list.get(i)[0],
				list.get(i)[1]);
		eval = subBoard.getBlack() -
				subBoard.getWhite();
		if(subBoard.player == 1){
			if(eval > maxvalue){
				maxvalue = eval;
				masu[0] = list.get(i)[0];
				masu[1] = list.get(i)[1];
			}
		}else{
			if(eval < minvalue){
				minvalue = eval;
				masu[0] = list.get(i)[0];
				masu[1] = list.get(i)[1];
			}
		}
	}
	return masu;
}

if (state.getTurn() == 1) {
	//自身が最初の１手の場合に最善手を選ぶようにする
	int num = rnd.nextInt(2);
	for (int i = 0; i < list.size(); i++) {
		subBoard = state.copy();
		subBoard.put(list.get(i)[0],
				list.get(i)[1]);
		eval = -alphabeta(subBoard, 5,
				-Integer.MAX_VALUE,
				-Integer.MIN_VALUE);
		v1.add(new Value(list.get(i)[0],
				list.get(i)[1], eval));
```

```
                }
                Collections.sort(v1, new ValueComparator());

                masu[0] = v1.get(0 + num).getX();
                masu[1] = v1.get(0 + num).getY();

            } else {
                //自分の候補手とその評価値をArrayListのv1 に格納する
                for (int i = 0; i < list.size(); i++) {
                    subBoard = state.copy();
                    subBoard.put(list.get(i)[0],
                                list.get(i)[1]);
                    eval = -alphabeta(subBoard, 5,
-Integer.MAX_VALUE,
                                -Integer.MIN_VALUE);
                    v1.add(new Value(list.get(i)[0],
list.get(i)[1], eval));
                }
                //評価値の順に並べ替えていく
                if(this.color == 1){
                    Collections.sort(v1,
                                new ValueComparator());
                }
                else{
                    Collections.sort(v1,
                                new ValueComparatorR());
                }

            //前の手番の相手の候補手とその評価値をArrayListのv2 に格納する
                movecheck(subPreBoard, list);
                for (int j = 0; j < list.size(); j++) {
                    subPreBoard = state.getPreBoard().copy();
                    subPreBoard.put(list.get(j)[0],
                                list.get(j)[1]);
                    eval = -alphabeta(subPreBoard, 5,
                                -Integer.MAX_VALUE,
                                -Integer.MIN_VALUE);
                    v2.add(new Value(list.get(j)[0],
                                list.get(j)[1], eval));
                }
                //評価値の順に並べ替えていく
                if(this.color == 1){
                    Collections.sort(v2,
                                new ValueComparatorR());
                }
                else{
                    Collections.sort(v2,
```

```
                                        new ValueComparator());
                }


                //相手の評価値の高さによって自分の手を同じ高さの評価値の候
補手の座標を返す
                for (int h = 0; h < v2.size(); h++) {
                        if ((v2.get(h).getX() == state.log[0])
                        && (v2.get(h).getY() == state.log[1]))
                                counter = h;
                }

                if (counter >= (v1.size() - 1)) {
                        int num = rnd.nextInt(2);
                        masu[0] = v1.get(v1.size() - 1 +
                                        range[num]).getX();
                        masu[1] = v1.get(v1.size() - 1 +
                                        range[num]).getY();

                }else if(counter == 0){
                        int num = rnd.nextInt(2);
                        masu[0] = v1.get(0 + num).getX();
                        masu[1] = v1.get(0 + num).getY();

                }else {

                        int num = rnd.nextInt(3);
                        masu[0] = v1.get(counter +
                                        range[num]).getX();
                        masu[1] = v1.get(counter +
                                        range[num]).getY();

                }

        }
        return masu;
}

/**
 * alphabetaメソッド
 * @param state 現在の盤面をコピーするための引数
 * @param depth 探索する深さを入力するための引数
 * @param alpha 最高値を代入するための引数
 * @param beta 枝切りを行うための引数
 * @return 候補手の評価値
 */
public int alphabeta(GameState state, int depth,
```

```
                int alpha, int beta){
        GameState sub;
        int value = 0;
        ArrayList<int[]> list = new ArrayList<int[]>();

        if (depth == 0)
                return evaluate(state);

        if (state.checkPass()) {
                sub = state.copy();
                sub.pass();
                value = -alphabeta(sub, depth - 1,
                                -beta, -alpha);
                return value;
        }

        movecheck(state, list);
        for (int i = 0; i < list.size(); i++) {
                sub = state.copy();
                sub.put(list.get(i)[0], list.get(i)[1]);
                value = -alphabeta(sub, depth - 1,
                                -beta, -alpha);
                alpha = Math.max(alpha, value);
                if (alpha >= beta) {
                        return alpha;
                }
        }

        return alpha;
}

/**
 * evaluateメソッド
 * @param state 現在の盤面をコピーするための引数
 * @return 盤面の評価値
 */
public int evaluate(GameState state) {
        int value = 0;
        for (int x = 0; x < 8; x++) {
                for (int y = 0; y < 8; y++) {
                        value += state.data[x][y] *
                                eval.values[x][y];
                }
        }
        return value;
}
```

```
	/**
	 * movecheckメソッド
	 * 候補手を探すメソッド
	 * @param state 現在の盤面をコピーするための引数
	 * @param list 格納するためのArrayListを入力するための引数
	 */
	public void movecheck(GameState state,
			ArrayList<int[]> list) {
		list.clear();
		// どこに石を置けるかチェックする
		for (int x = 0; x < 8; x++) {
			for (int y = 0; y < 8; y++) {
				// 石があるところは飛ばす
				if (state.data[x][y] != 0)
					continue;

				if (state.canreverse(x, y)) {
					int pos[] = { x, y };
					list.add(pos);
				}
			}
		}
	}
}
```