卒業研究報告書

題目

モンテカルロ法を用いたオセロ AI

指導教員

石水　隆　講師

報告者

09–1–037–0030

藤田　竜平

近畿大学理工学部情報学科

平成 28 年 1 月 29 日提出

概要

オセロとは、二人零和完全情報ゲームである。オセロやニップ、チェス、将棋などの二人零和完全情報ゲームは理論上は最善手が存在する。しかし多くの場合、可能な局面数が膨大であるため最善手を求めることは出来ない。オセロもゲーム展開の数は 10^{53} 通りほどあると言われており、未だに最善手は求められていない。オセロの対戦の場となるのは、8×8 のマスの目であり、白と黒の石を使いどちらが盤上に多く置くことができるかが勝敗を決する。オセロのルールは至って単純であり、自分の使用する石の色で相手の使用している石を挟み、自分の使用している石の色にひっくり返すという作業を繰り返していくというものである。ゲームは単純だが、極めようとすると一生かかる言われている実に奥の深いゲームと言われている。

本研究では、オセロプログラムを作成した。作成の際に取り入れたのがモンテカルロ法 [6] である。モンテカルロ法とは、乱数を用いたシミュレーションを複数回行って近似値を求める計算手法である。本研究ではモンテカルロ法を用いたオセロ AI がどの様な動きをするか、その有用性を検証する。

目次

1 序論

1.1 本研究の背景

オセロや将棋、囲碁、チェス、ニップ等のボードゲームは二人零和有限確定完全情報ゲームである。二人零和有限確定完全情報ゲームとは、以下の 4 つの条件を満たすゲームである。

- 二人・・ゲームを行うプレイヤーが二人である。
- 零和・・ゲーム上プレイしている全プレイヤーの利得の合計が常に 0、または個々のプレイヤーの指す手の組合せに対する利得の合計が全て一定の数値（零和）となるゲーム。
- 有限・・そのゲームにおける各プレイヤーの可能な手の組合せが有限であるゲーム。
- 確定・・プレーヤーの着手以外にゲームに影響を与える偶然の要素が入り込まないゲーム。
- 完全情報・・各プレイヤーが自分の出番において、これまでの各プレイヤーの行った選択について全ての情報を知ることが出来るゲーム。

二人零和有限確定完全情報ゲームは、その性質上から解析しやすく、ゲーム理論において様々な研究が行われてきた。

1.2 二人零和有限確定完全情報ゲームの既知の完全解析の結果

二人零和有限確定完全情報ゲームは各プレイヤーが最善手を打ち続ける事によって「先手必勝」「後手必勝」「両者引き分け」が最初から決まっている。今現在の時点では、数多くあるボードゲームは解析する局面での数が多すぎる為、完全解析には至っていない。オセロに関しては可能な局面数が 10^{53} 通りあり、現在の計算機の性能を越えている。一方で、可能な局面数が少ないボードゲームでは完全解析されている物もある。連珠は二人のプレイヤーが最善手を打った場合、47 手で先手が勝利する [7]。チェッカーは二人のプレイヤーが最善手を指した場合、引き分けとなる [8]。局面数が多いボードゲームでは、ゲーム盤を本来より小さくした場合の完全解析が行われている。オセロではサイズが 6×6 の盤の場合、二人のプレイヤーが最善手を打つと、16 対 20 で後手の勝利となる [9]。将棋では、サイズを 3×4 に縮小し、駒の種類を 4 種類に減らした「どうぶつしょうぎ」に対して、二人のプレイヤーが最善手を指した場合、78 手で後手の勝利となることが判明している [10]。

1.3 完全解析されていない二人零和有限確定完全情報ゲームに対する手法

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現時点では困難である。しかし、完全解析出来ていないゲームに対しても、局面の評価値計算、定石データベース、モンテカルロ法、一定手数の先読み、終盤での必勝先読みと完全読み等を用い、より有利だと思われる手を選択することが可能になる。オセロに関しては昔から様々な研究が行われてき、今では対人だとまず負けることのないレベルまできているが、完全解析には至っていない。しかしこれはプログラミングが問題なのではなく、今現在の計算機の問題であるというのが大方の見方であるとされている。

1.4 既知の結果

本節ではコンピュータオセロに関する既知の結果を述べる。手法として定石データベースのみを用いたLogistello[12] がある。Logistello は 1997 年に当時世界チャンピオンの村上健八段に勝利しており、その強さを証明している。しかし、Logistello は定石データベースのみを使用するため、定石以外の手を打たれた場合には対応できない。その他には Zebra[11] や Edax[17] などもある。これら 2 つのオセロ AI にも book と呼ばれる序盤、中盤の定石データや対戦データベースを使用しているが、中盤の先読みと終盤の完全読みを行っている。特に Edax はマルチコア対応で高速処理が可能となっているが、book 非使用時は勝率が下がるというのが一般的な見方である [2]。

1.5 本研究の目的

モンテカルロ法は主に囲碁の AI 作成の際に使用されているが、オセロ AI ではあまり使用されていない。囲碁 AI を作成する際にモンテカルロ法は絶大な効果を発揮しておりこの手の手法は他のゲームにも応用が利くと考えられる。そこで本研究では、モンテカルロ法を用いたオセロ AI を作成し、その有用性を検証する。

1.6 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 章でオセロのルールと定石について述べる。続く第 3 章でオセロ AI の作成に用いられてる手法について述べる。第 5 章にて作成したオセロ AI について述べ、第 6 章にて実験結果を報告、考察する。第 7 章にて研究の結論、今後の課題を示す。

2 オセロについて

2.1 オセロのルール

オセロは 8 × 8 のマスの盤面を用いてゲームを行う。初期配置として、盤面の中心点 2 × 2 のマスに白と黒の石を交差させた状態で並べておく。図 1 に初期配置を示す。

プレイヤーは空きマスのいずれかに自分の使用している色の石を置いていく。ただし、石を置けるマスには条件があり、自分の使用している色の石で相手の使用している色の石を挟める場所でないと置けない。打つ場所が無い場合は石を打たずにパスをする。石を置いた時、挟んだ相手側の石を裏返し自分の使用している石の色にする。双方石が置けなくなった時点でゲーム終了とする。最終的な勝敗は石の色の数の多い方こ勝ちとなる。同数の場合は引き分けとなる。

2.2 オセロの完全解析

オセロは 60 手以内に必ず勝敗が決する。従って 60 手先まで先読みし全てを解析することが出来ると最善の手を打つことが可能である。しかし先読みの数が増えるにしたがい、計算の量が先読み数の指数乗に比例して増加する為今現在完全解析をするのは不可能である。しかし、盤面のサイズを小さくしたものについては可能な局面数が少なくなり、完全解析が可能となる。6 × 6 の盤面のオセロは 1994 年にイギリスの研究者 Feinstein によって後手必勝であり、最善手を打った時、先手は最大 16 個しか取れないということが証明され

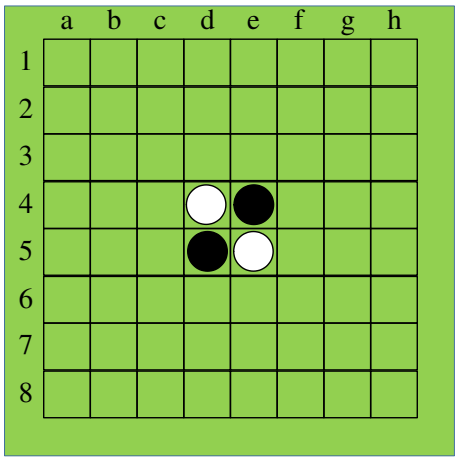

図 1　初期配置

ている。[9]

2.3　序盤、中盤の定石

オセロは完全解析はされていないが、序盤においてはどのような手が有利になるかは、定石として確立している。代表的な序盤の定石には、縦取り兎定石、斜め取り牛定石、並び取り鼠定石がある [13]。それぞれの盤面例を図 2,3,4 に示す。

局面が様々に変化する中盤においても定石がいくつか確立されている。代表例として中割り、引っ張り等がある。[13] 中割りは「周りが全て石に囲まれている石のみ返す」という事を意味し、相手の打てるマスを増やさないための手筋である。引っ張りは壁を作り、相手はこちらの壁を崩さなければ石が置けない状態を作ることで、相手の石を誘導したいときに用いる手筋である。中割りの手筋を図 5,6、引っ張りの手筋を図 7,8 に示す。

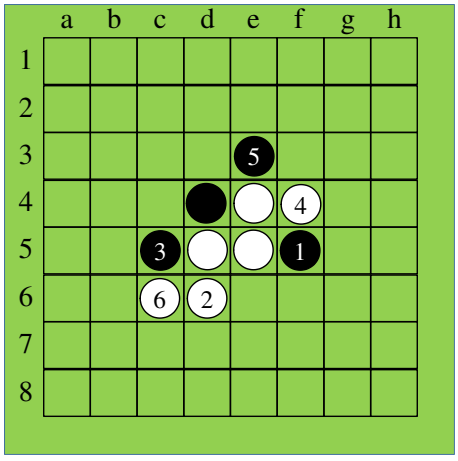

図 2　縦取り兎定石

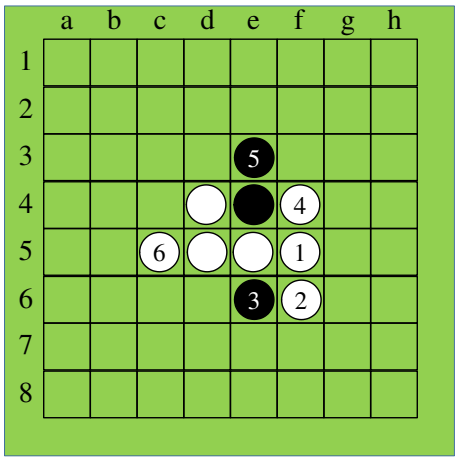

図 3　斜め取り牛定石

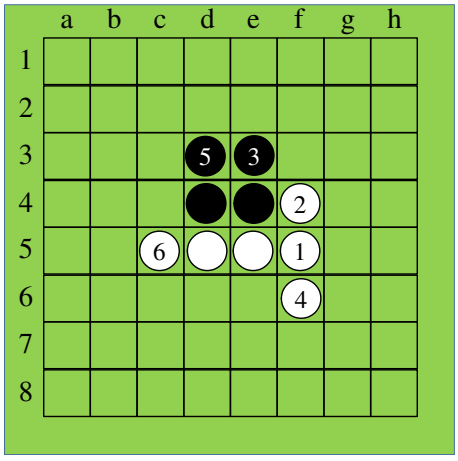

図 4　並び取り鼠定石

3　オセロ AI の一般的手法

本章では、オセロ AI が用いてる一般的な手法について述べる。

3.1　定石データベース

定石データベースとは、オセロの定石をデータベース化し、各局面で有効な定石があればそれに従って打つという手法である。定石データベースを使用することで強いオセロ AI の作成が可能となる。しかし、相手があえて定石以外の手を打つなどした時にデータベースに無い局面が出てきた時には定石データベースは使えない。代表例として Edax[11]、WZebra[12] 等のオセロ AI が存在する。これらの AI は盤面に置かれた駒のパターンによって評価値を算出しているものである。他にも過去のデータを用いて遺伝的アルゴリズムによって

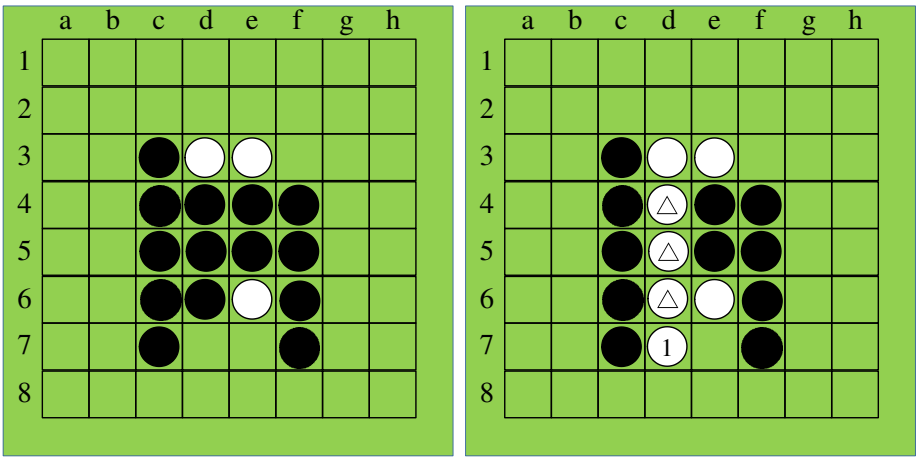

図 5　中割り

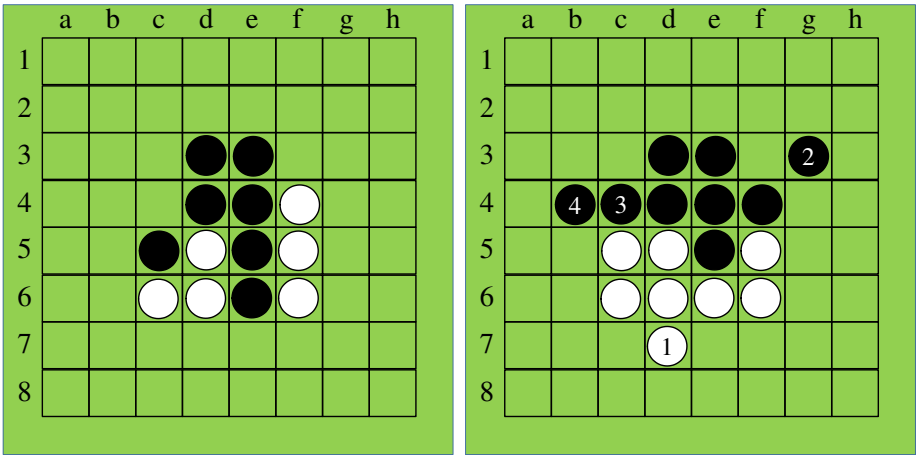

図 6　引っ張り

評価値のバランス調整がされている AI も存在している。

3.2　評価関数

評価関数は評価関数を用いて現在の局面、または数手先の局面を評価する手法である。一般的に初心者でも組みやすいとされているのが盤面評価値である。盤面評価値とは、盤面 1 マスずつに重みをつけてある局面をその値で評価する。ただし、盤面評価値は盤面の重みだけで見るため全体の局面を見ることができないという問題があり、強いオセロ AI を作成することは難しい。

また、ある局面の評価値を求める評価関数は、現在の局面のみを考慮する現局面評価と、数手先の局面を先読みし、先読みした局面に対して現局面評価を行い、その評価値を元に現在の局面の評価値を求める先読み局面評価の 2 つに分類される。

現局面評価の評価関数の計算に用いられる評価基準については、先に記述した盤面評価値、直線性、確定

石、駒数、候補数など様々なものが提案されている [6]。

3.3 モンテカルロ法

モンテカルロ法 [6] とは、乱数を用いたシミュレーションを複数回行って近似値を求める計算手法である。現在はオセロ AI ではあまり使用されておらず、主に囲碁 AI の作成の際使われている手法である。

モンテカルロ法の詳細については第 4 章で述べる。

3.4 先読み

先読みとは、可能な範囲で一定数の先の手を読み、その手から作られる局面の評価値を求め、最も評価値が高い手を採用することである。一般に先読み手数を増やすにつれ評価関数の精度は上がり強い 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、探索範囲の縮小等の工夫が必要である。

3.5 必勝読みと完全読み

オセロは終盤に近付くにつれ、勝敗が決定するまでの手数が少なくなり選択できる手も限定される為、勝敗が決するまで読みきることが容易となる。終盤での読みの手法として必勝先読みと完全読みがある。必勝読みとは、ゲームの終盤で勝敗のみを読み切ることで必勝の手を打つことを言う。完全読みとは、ある局面で得られるすべての局面を読み、最も良いとされる手を打つことを言う。

必勝先読みの方が計算時間が少なくて済む為、一般にまず必勝読みで勝ちを確立させた上で、残り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。

4 モンテカルロ法

モンテカルロ法 [6] とは、乱数を用いたシミュレーションを複数回行って近似値を求める計算手法である。ゲーム AI の分野だけでなく、物理的な現象の推測や経済的な分野等様々に広く用いられている。ギャンブルで有名なモナコの地名が由来となっている。シミュレーションを行うことで近似解を求める仕様上、シミュレーションさえ行える問題ならばどれに対しても適用出来る範囲の広さが長所である。

5 本研究のオセロ AI

本研究では、モンテカルロ法を用いて着手を決定するオセロ AI(以下 montAI) を Java を用いて作成した。その性能を検証するために、ランダムに着手を決定するオセロ AI(以下 randomAI) も作成した。本研究で作成した montAI および randomAI のソースコードを付録 A に示す。

5.1 本研究のモンテカルロ法の手順

montAI で用いているモンテカルロ法の手順を示す。1. 現在の局面での候補手を探す。2. 各候補手に対して一定回数のプレイアウトを行う。プレイアウトとは、現在の局面から、ランダムに着手を行って最終局面まで進め、その勝敗を求めることである。このとき、プレイアウトで得られた勝率をその候補手の評価値とする。3. 候補手の中で最も評価の高い手を選択する。

5.2 モンテカルロ法を用いたオセロ AI

本節では、montAI についてその詳細を述べる。

montAI の中心となるクラスが class mont である。class montAI には、表 1 に示す 2 つのメソッドがある。

void bestmove(ConsoleBoard board) メソッドはモンテカルロ法に基づいた最適手を探索するメソッドである。引数 board のメソッドを用いて現在の盤面状況から着手可能手を得た後、着手可能手に対し vertualGame() メソッドで仮想対局をプレイアウト数分行い、全評価値の和を取る。評価値の和の最も高い手の座標を変数 dedicate に格納し、Board クラスの bestmove() を用いて着手する。

int vertualGame(ConsoleBoard board nB,Point montPoint) メソッドは着手手を打った後、終局まで打ち続けるシミュレーションを行うメソッドである。仮想ボードを作成し、ランダムに打つ AI(randomAI クラス) を利用して終局まで打つのを複数回行う。

private int trialN = でプレイアウト数の変更を行う。

表 1　montAI クラスのメソッド

メソッド	内容
void bestmove(ConsoleBoard board)	最適手探索
int vertualGame(ConsoleBoard board nB,Point montPoint)	シミュレーション

6 結果・考察

モンテカルロ法を使用したオセロ AI の有能性を検証するため、プレイアウト回数を 500 回とした montAI(500) とランダムで打つオセロ AI(以下 randomAI) との対戦、および montAI(500) とプレイアウト数を 100 回にした montAI(100) との対戦をそれぞれ 100 回ずつ行った。表 2, 表 3 に実行結果を示す。

表 2　AI 同士の対戦結果 (試行回数 100 回)

先手	後手	先手勝ち	後手勝ち
montAI(500)	randomAI	83	17
randomAI	montAI(500)	14	86

表 3　AI 同士の対戦結果 (試行回数 100 回)

先手	後手	先手勝ち	後手勝ち
montAI(500)	montAI(100)	73	27
montAI(100)	montAI(500)	21	79

表 2 より、先手後手共に montAI(500) が 8 割以上の勝率を得ることが示される。表 3 より、プレイアウトの回数が大きいほど、勝率が高くなることが示される。本研究で作成したモンテカルロ法 AI はランダムに対

し 8 割以上の勝ちを収めており、当初の目標としていた勝率に達することが出来た。しかしランダム相手に勝率 8 割というのは、決して強いとは言えない結果である。

7　結論・今後の課題

本研究ではモンテカルロ法を使用したオセロ AI を作成した。本研究で作成したプログラムは randomAI に対して勝率 8 割以上と当初の目標に達することが出来た。しかし randomAI 相手に 8 割程度ではまだまだ良い AI とは言えず、本プログラムは改善の余地がまだまだあると思われる。

今後の課題としては、プレイアウト数をもっと増やした AI で対戦させる事でモンテカルロ法の有用性を検証する事が挙げられる。プレイアウト数を増やす事で実行時間が膨大になる事が予想される為、並列化により処理を高速化させ、プレイアウト回数を増やしても実行時間がかからないようにすることも挙げられる。また、AI の対戦レベルを設けてゲームとしての面白さを加えることも課題の 1 つである。

謝辞

本研究書を作成するにあたり、担当して下さった担当の石水隆先生には大変お世話になりました。ご迷惑をおかけしたことのお詫び、及び大変熱心に最後まで本研究のご指導してくれましたことへの感謝の気持ちをこの場を借りて心より申し上げます。

8 参考文献

参考文献

1) 上野早也香, モンテカルロ法を用いたオセロプログラム, 近畿大学理工学部情報学科　2014 年度 卒業研究報告書,(2015) http://www.info.kindai.ac.jp/ takasi-i/thesis/$2014_11 - 1 - 037 - 0017_{SU}eno_thesis.pdf$

2) Seal Software:リバーシのアルゴリズム, 工学社 (2007).

3) 保田和隆:オセロ・リバーシプログラミング講座 (2011) http://uguisu.skr.jp/othello/

4) 日本オセロ連盟,http://www.othello.gr.jp/biginner/$s_01.html$, オセロ講座, 初心者, 2002

5) 大筆豊, オセロプログラムの評価関数の改善について, 研究報告ゲーム学 (GI),Vol.2003-GI-011,pp.15-20, 情報処理学会,(2004). http://id.nii.ac.jp/1001/00058554/

6) 美添一樹, 山下宏, 松原仁:コンピュータ囲碁-モンテカルロ法の理論と実践-, 共立出版 (2012)

7) Janos Wagner and Istvan Virag Solving renju,ICGA Joumal Vol.24,No.1,pp.30-35,(2001),http://www.sze.hu/ gtakacs/downlord/wagnervirag 2001.pdf

8) Jonathan Schaeffer, Neil Burch, Yngvi Bjorsson, Akihiro Kishimoto, Martin Muller, Robert Lake, Poul Lu, and Steve Suphen, Checkers is sloved, Science Vol.317,No.5844, pp.1518-1522,(2007).

9) Joel Feinstein, Amenor Wins World 6 × 6 Chanpionship!, Forty billionnoted under the tree (July 1993),pp.6-8,British Othello Federation's newsletter,(1993).

10) 田中哲朗,[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.3pp.1-8, 情報処理学会,(2009),http://id.nii.ac.jp/1001/00062415

11) Richard Delrme,Othello programing,(2012),http://abulmo.perso.neuf.fr/index.htm

12) Gunnar Andersson,Wzebra,(2006),http://rabagast.se/othello/

13) 谷田邦彦, 図解早わかりオセロ　これが必勝のコツだ！！, 日東書院 (2003).

14) 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 (GI),Vol.2009-GI-22,No.7,pp.1-8, 情報処理学会 (2009), http://id.nii.ac.jp/1001/00062419

15) 村山要司, 楽しく学べる Java ゲーム・アプレット, 工学社 (2002).

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

17) T.Ishii,MasterReversi, 他アプリとの対局結果 http://homepage2.nifty.com/$t_ishii/mr/gameresult.html$, 2007.

18) Michael Buro,Logistello,https://skatgame.net/mburo/log.html,1997

19) 稲田康平, オセロにおけるモンテカルロ法プレイヤの性能調査, 高知工科大学 情報学群 平成 25 年度 学士 学位論文,(2014), http://www.kochi-tech.ac.jp/library/ron/2013/2013info/1140296.pdf

付録 A　付録

本研究で作成した Java プログラムソースを以下に示す。

```
import java.util.*;

public class GameState extends Observable{


        int data[][];
        int turn;
        int player;
        int black;
        int white;

        public GameState(){


                data = new int[8][8];
                data[3][3] = 1;
                data[3][4] = -1;
                data[4][3] = -1;
                data[4][4] = 1;


                turn = 0;
                player = 1;
                black = 2;
                white = 2;
        }



        public boolean put(int x, int y){すでに駒があるところには置けない
                //
                if(data[x][y] != 0){
                        return false;
                }リバースできないところには置けない


                //
                if(reverse(x,y,true)==false){
                        return false;
                }駒を置く



                //
                data[x][y] = player;
                player *= -1;
                turn++;
```

```
        countDisc();

        setChanged();
        notifyObservers();

        return true;
}



public boolean reverse(int x,int y, boolean doReverse ){
        int dir[][] = {
                        {-1,-1}, {0,-1}, {1,-1},
                        {-1, 0},         {1, 0},
                        {-1, 1}, {0, 1}, {1, 1}
        };


        boolean reversed = false;


        for(int i=0; i<8; i++){隣のマス
                //
                int x0 = x+dir[i][0];
                int y0 = y+dir[i][1];
                if(isOut(x0,y0) == true){
                        continue;
                }
                int nextState =data[x0][y0];
                if(nextState == player){
                        System.out.println("Next state is player: " +x0 +","+ y0);
                        continue;
                }else if(nextState == 0){
                        System.out.println("Next state is null: " +x0 +","+ y0);
                        continue;
                }else{
                        System.out.println("Next state is enemy: " +x0 +","+ y0);
                }隣の隣から端まで走査して、自分の色があればリバース

                //
                int j = 2;
                while(true){

                        int x1 = x + (dir[i][0]*j);
                        int y1 = y + (dir[i][1]*j);
                        if(isOut(x1,y1) == true){
                                break;
                        }自分の駒があったら、リバース

                        //
                        if(data[x1][y1]==player){
                                System.out.println("Player cell!: " +x1 +","+ y1);
```

```
                                if(doReverse){
                                        for(int k=1; k<j; k++){
                                                int x2 = x + (dir[i][0]*k);
                                                int y2 = y + (dir[i][1]*k);
                                                data[x2][y2] *= -1;
                                                System.out.println("reverse: " +x2 +","+ y2);
                                        }
                                }
                                reversed = true;
                                break;
                        }空白があったら、終了

                        //
                        if(data[x1][y1]==0){
                                break;
                        }

                        j++;

                }

        }

        return reversed;
}



public boolean canReverse(int x, int y){
        return reverse(x, y, false);
}

public boolean isOut(int x, int y){
        if(x<0 || y<0 || x>=8 || y>=8){
                return true;
        }
        return false;
}

public boolean checkPass(){コピーデータの全升目に対して、リバースできるかチェック

        //
        for(int y=0; y<8; y++){
                for(int x=0; x<8; x++){すでに駒があるところはチェックしない

                        //
                        if(data[x][y] != 0){
                                continue;
                        }リバースできる（した）とき、元に戻して

                        //を返すfalse
                        if(canReverse(x,y) == true){
                                return false;
```

```
                                }

                        }
                }

                return true;
        }

        public void countDisc(){

                black = 0;
                white = 0;

                for(int y=0; y<8; y++){
                        for(int x=0; x<8; x++){
                                if(data[x][y] == 1){
                                        black++;
                                }else if(data[x][y] == -1){
                                        white++;
                                }
                        }
                }
        }


}


import java.util.*;


public class randomAI {

        int color;      //BLACK or WHITE

        public randomAI(){
                color = -1;
        }

        int[] decide(GameState state){

                ArrayList<int[]> array = new ArrayList<int[]>();盤面の空マスを置けるかチェック

                //
                for(int y=0; y<8; y++){
                        for(int x=0; x<8; x++){すでに駒があるときはパス

                                //
                                if(state.data[x][y] != 0)
                                        continue;置けるマスのとき、候補として記憶

                                //
                                if(state.canReverse(x, y) == true){
```

```
                                        int pos[] = {x,y};
                                        array.add(pos);
                                }

                        }
                }ランダム選択

                //
                if(array.size() <= 0){
                        int pos[] = {-1, -1};
                        return pos;
                }
                Random rnd = new Random();
                int index = rnd.nextInt(array.size());

                return array.get(index);
        }

}

import java.util.*;モンテカルロ法


//
public class montAI extends Observable{
        class Move extends Observable
        {

                public int eval =0;
                public bestmove()

                super(0,0);
        }
        public bestmove(int x,int y int e)
        {
                super(x,y);
                eval = e;
        }
};

class randomAIPlayer implements Player
{

        private AI randomAi = null;

        public randomAIPlayer()
        {

                  randomAi = new randomAI();

        }

        public void onTurn(ConsoleBoard board) throws GameoverException
```

```
        {

                randomAi.bestmove(board);

                if(board.isGameover()) throws new GameoverException();
        }
};プレイアウト数

//
private int trialN=100;盤面操作決定


//
public void bestmove(ConsoleBoard board)
{
        BookManager book = new BookManager();
        Vector<point> movables = new book.find(board);


        if(movables.Empty())
        {パスをする

//
        board.pass();

        return;
}


 if(movables.size()=1)
 {打てる箇所が一ヶ所なら探索は行わない

        //
        board.move((Point) movables.get(0));
        return;
}モンテカルロが試行対象とする一手とボードデータを受け取り、ランダム


//同士で最後まで打ち切るAI
private int vertualGame(ConsoleBoard nB,Point montPoint)


{
        ConsoleBoard nBoard = new ConsoleBoard();
        nBoard = nB;
        System.out.println();
        nBoard.print();評価値
        //
        int evaluation=0;
        Player[] player new Player[2];どちらのターンか知る
        //
        int current_player;
        int fast_trun = nBoard.getCurrentColor();
```

```
        fast_turn = 1;
        System.out.println色の確認(""+fast_turn);
        player[0] = newrandomPlayer黒();
        player[1] = newrandomPlayer白();
        if(fast_trun =1)
        {

                current_player=0;
        }
        else
        {

                current_player=1;
        }最初の一回のみ引数で受け取った座標を打つ。以降ランダム
        //
        nBoard.move(montPoint);プレイヤー交代
        //
current_player = ++current_player2;終了までループ

        //
        while(turn)
        {

                try{
                        player[current_player].onTurn(nBoard);
                 }
                 catch(doException n)
                 {
                        do
                        {
                                System.out.println("do");
                                nBoard.do();nBoard.do();
                        }while(nBoard.getMovablesPos().isEmpty());
                        contenue;
                 }
                 catch(ExitException n)
                 {
                        n.printStackTrace();
                 }
                 catch(BameOverException n)
                 {

                        System.out.println終了("");
                        System.out.print(nBoard.countDisc(Disc.BLACK));
                        System.out.print(nBoard.countDisc(Disc.WHITE));
                        if(fast_turn=1){
                                evalution = nBoard.countDisc(Disc.WHITE)
                                nBoard.countDisc(Disc.BLACK);
                        }
```

```
        else
        {


                evalution = nBoard.countDisc(Disc.BLACK)
                nBoard.countDisc(Disc.WHITE);
        }


        break;
}


return evalution;
        }
```