卒業研究報告書

題目

盤面評価値を用いたオセロプログラム

指導教員

石水　隆　講師

報告者

10-1-037-0024

岸本　浩一

近畿大学理工学部情報学科

平成 27 年 1 月 31 日

概要

オセロとは、8×8 の盤面で構成され、白と黒の 2 つの駒で交互に駒を打ち、相手の駒をはさんで裏返し盤面がいっぱいになった時に駒数が多いほうが勝利するという単純なゲームであり、その反面奥が深く知能ゲームの要素を持ち世界中に広まった。また、現在の技術を駆使してさえ完全解析されていないゲームの 1 つである。現在のコンピューターに思考させることは不可能なため、思考手順を教え、その手順に従って最善の手を計算させる。

本研究では、オセロにおける盤面の評価関数を変化させた時、勝率がどのように変わるかを観測し最適な組み合わせを求める。

目次

1.序論

1.1 二人零和有限確定完全情報ゲーム

二人零和有限確定完全情報ゲームは双方最善手を打った場合、先手勝ち、後手勝ち、引き分けのどれになるかはゲーム開始時点で決定しており、理論上、全ての可能な局面を解析することができれば最善の手を打つことができる。しかし多くのボードゲームでは、可能な局面の総数が極めて大きいため、完全解析を行うことは不可能である。例を挙げれば、オセロが 10^{20} 通り、チェスが 10^{50} 通り、将棋が 10^{69} 通り、囲碁が 10^{170} 通り程度あるとされており、現在の計算機の性能を越えている。

一方、可能な局面数が少ないゲームでは完全解析されているものもある。連珠は双方最善手を打った場合、47 手で先手が勝つ[16]。チェッカーは双方最善手を指すと引き分けとなる[17]。

局面数が大きいゲームについては、ゲーム盤をより小さいサイズに限定した場合の解析も行われている。囲碁については、サイズ 4x4 の囲碁は双方最善手を打つと持碁(引き分け)[18]、5x5 の囲碁は黒の 25 目勝ちとなる[19]。オセロについては、盤面の数を 6×6 にした場合後手必勝であり、また後手が最善手を打った時、先手は最大でも 16 個しか自石を取れないことが J.Feinstein により証明されている[5]

1,2 オセロ

オセロとは、8×8 のマスを使用する。各マスには、横に a～h、縦に 1～8 の座標があり、自石で相手石を挟めるマスに石を置くことができ挟んだ石は反転させ自石となる。

先手、後手交互に打っていき打てるマスがない時は、パスとなり相手に石を打つ権利が譲られる。盤面が全て埋まるまた、先手、後手が共に石を挟めなくなった時点でゲーム終了となり、自石の数が多いプレイヤーが勝利同数の場合は引き分けとなる。

1.3 コンピュータゲームの人工知能

コンピューターにおける人工知能とは、コンピューターのゲームにおいて、NPC の振る舞いに知能を出す技法である。主な技法は AI の既存技術を活用したものである。しかしゲーム AI と呼ぶ場合、制御理論などの全般の技法を含む様々なアルゴリズムを指して使われることが多い。

1,4 オセロに関する既知の結果

本節では、オセロに関する既知の結果を示す。

1,4,1 オセロの完全解析

オセロは 1 ゲームにつき最大 60 手順しかないことから対戦型ゲームのなかでも手順が少ないゲームであるが、白黒の 2 通り、それ以外 60 マスは白黒空きの 3 通り、可能である局面の組み合わせは $2^{4}*3^{60}$ $6{,}78*10^{19}$ 通り存在する。このためオセロは現時点でスーパーコンピューターを駆使してもなお完全解析されていない。しかし盤面の数を 6×6 にした場合後手必勝であり、また後手が最善手を打った時、先手は最大でも 16 個しか自石を取れないことが証明されている[7]。

1,4,2 オセロの定石

前節で述べたが、オセロは完全解析がされていなく特に序盤においては、どの手も最善となるかを決定することはできない。しかし、序盤においてはどのような手が有利になりやすいかは定石として確立されている。代表的な序盤の定石には、兎定石、牛定石、鼠定石があり、兎定石は縦取り、牛定石は横取り、鼠定石は、並び取りである。これらの定石を図 2,3,4 に示す。また定石には中盤においても定石があり代表的には中割り、引っ張りがある。中割りとは、周りが全て石に囲まれている石を返すことで、相手の打てるマスを減らすための手である。また、引っ張りとは、自石で壁をつくり相手がその壁を崩さない限り石がおけないので、相手の石を誘導するときに用いられる手である。図 5,6 に中割、引っ張りの着手前と着手後の例を示す。

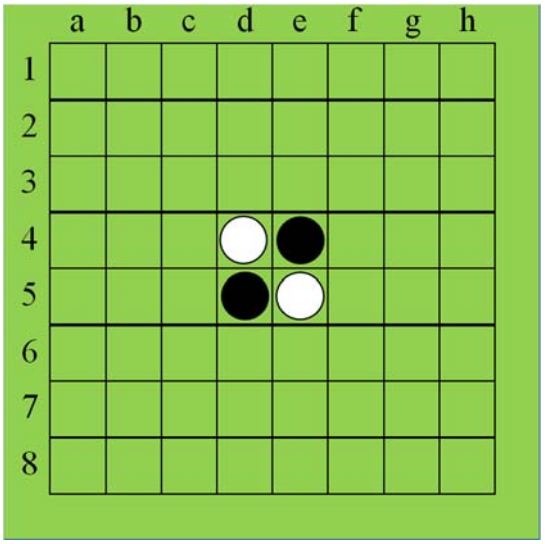

図 1 初期の盤面

序盤の戦略

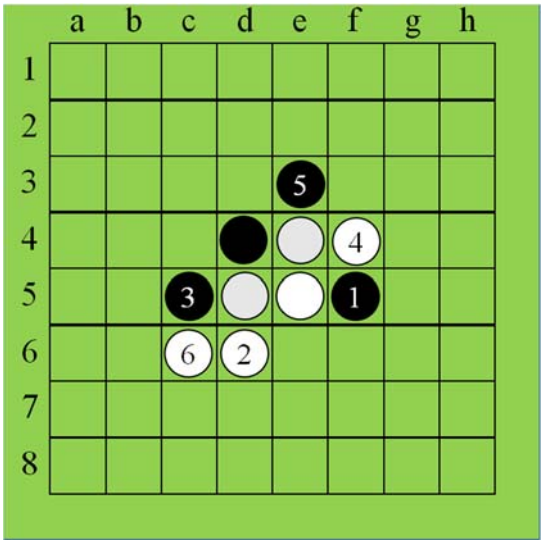

図 2 兎定石(縦取り)

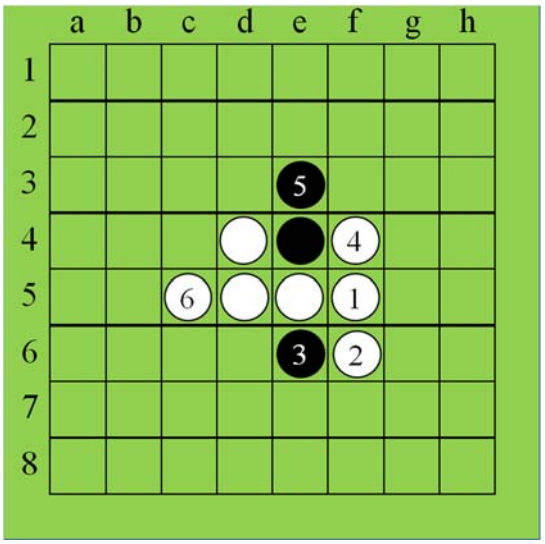

図 3 牛定石(横取り)

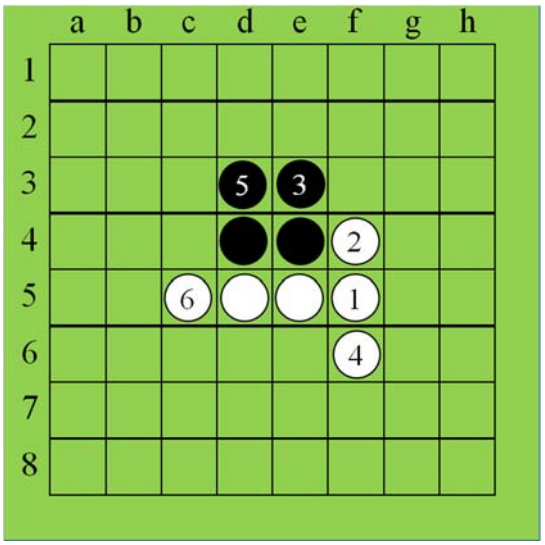

図 4 鼠定石(並べ取り)

中盤の戦略

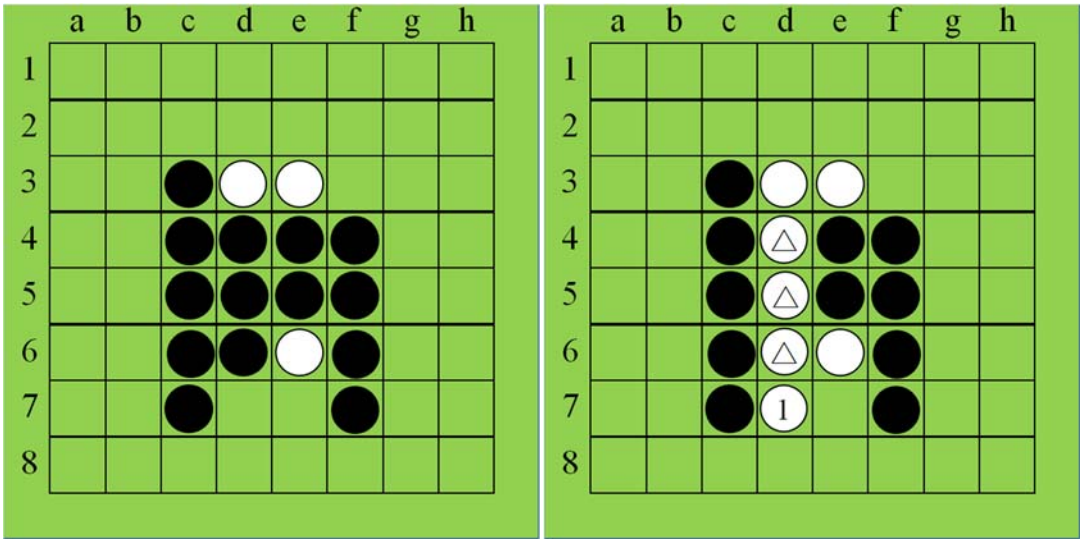

図 5 中割り

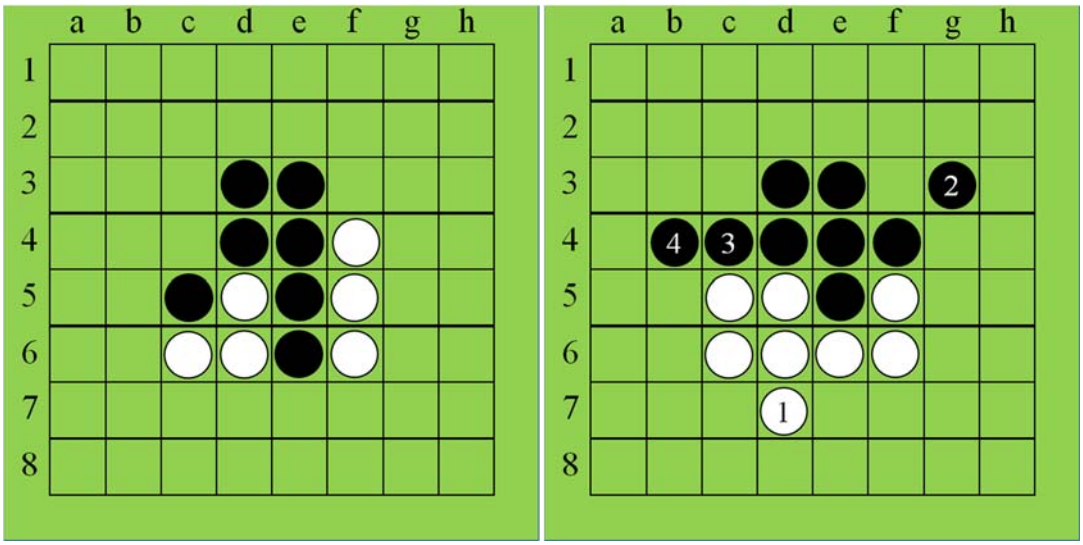

図 6 引っ張り

1,4,3 局面の評価

ゲーム中盤における局面の状況は様々あり、定石が存在しない局面もあり得る。また、序盤であっても相手が定石以外の手を打った場合も同じく定石が存在しない局面となる。そのような場合には最善の手を見つけるために用いられる 1 つとして局面の評価関数がある。この方法はある局面で盤上に置かれた石の並び方を入力する評価関数を設定し、その関数の値によって先手後手のどちらが有利を判定する。しかし、どのような評価関数を用いるのが最適かは未解決であり、様々な評価関数が提案されている。

1.4.4 先読み

現在の局面から数手先の局面を先読みし、それをもとに評価値を決めることによって評価関数の精度をあげられるもので、一般的には先読みをする手数を増やせば増やすほど評価関数の制度はあがる。また、最終局面まで先読みすることができれば勝敗を完全に決定できる完全な評価関数ができるが、先読み手数が増えるごとに可能な局面数は指数的にため勝負の序盤と中盤では完全先読みは不可能である。このため勝負の序盤と中盤では一定の数手先まで先読みし、得られた局面の各評価値をもとに評価値を求める手法が一般によく使われている。また、勝負の終盤では残り手が少ないのでその盤面から最終局面までの全ての先読みが可能である。現在の計算機機能では、残り 25～30 手程度でも最終局面までの完全先読みが可能である。

1.4.5 既存のオセロプログラム

オセロプログラムは、定石データベースの利用、局面の評価値計算、序盤・中盤で

の先読み、終盤の完全先読みのどれか、もしくは各手法の組み合わせを使用することが多い。

手法として定石データベースのみを用いたプログラムとして、Logistello[20]がある。Logistello は 1997 年には村上健八段(当時の世界チャンピオン)に勝利しており[11]、定石データベースを使うことでそれなりの強さのプログラムとなることが示された。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処できない。また、その他に Zebra[21]や Edax[22]などもある。これらについても 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 非使用の時は勝率が下がるというのが一般的な見方である[23]。

1.4.6 最新のオセロプログラム

最新のオセロプログラムには、BTD STUDIO 株式会社ゲームの HAYABUSA というオセロプログラム[6]がある。このプログラムではオープニングブック学習という機能がありオープニングブック学習とは、序盤の強さに良い影響を与え、評価関数は、盤面全体ではなく、各パターンに分けてそれぞれのパターンに評価をつけているため、プログラム上完璧ではなく、実際の盤面の良し悪しとずれることがある為、評価関数のエラーを補間するためにあり、常に負けた対局から学習し、思考を開始する前の段階で、既に打ったことのある対局の中で、最も良い と思われるラインを選択することが出来るというものである。

1.5 本研究の目的

オセロゲームは、完全解析がなされていないため、未だ最善手が存在しない。
そこで、ゲームを有利に進める為に盤面の評価をする評価関数を作成する。また、評価関数のパラメタに付加する重みを変化させ最も有効と思われる評価関数を求める。

1.6 本報告書の構成

本報告書の構成は次のとおりである。まず第 2 章でオセロプログラムの一般的な手法について説明する。第 3 章に本研究で用いた盤面の評価関数ついて説明する。第 4 章で評価関数を用いた対戦結果を示す。第 5 章で結論と課題を示す。

2 オセロプログラムの一般的な手法

前に述べた通りオセロプログラムの完全解析はできていないため、ある盤面において常に最善な手を選択することはできない。そこで本章では、オセロプログラムで用いられる一般的な手法について述べる。今現在強いと評価されているオセロプログラ

ムは、序盤は定石データベースに従って打ち、中盤、あるいは相手が定石から外れた手を打ったときの序盤では、一定手の数先読みを行い、先読み後の盤面に対しての評価関数を用いて評価値を求め、その値から打つ手を決定するというものである。そして終盤、残り手数 25～30 以下になると完全読みを行い、最善手を選択し打つというものである。

2.1 定石データベースと対戦データベース

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たもので、各盤面で有効な定石があればそれに従って手を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序盤では強いオセロプログラムとなる。しかし、相手があえて定石以外の手を打つなど、データベースに無い盤面になった時にはこの定石の手法は使えない。繰り返し対戦を行う場合は、それまでの対戦記録をデータベース化しておき参考にするという手法も考えられる、過去の対戦においてその手が最善手であったかどうかを対戦結果から判定し、それをデータベースにする。多く対戦することで、その手が有効かどうかより精度が高い判定をすることができると共に、対戦データベースを用いることで対戦経験が増えるにつれて強くなる人工知能型のオセロプログラムになる。対戦データベースを使うためには、事前に繰り返し対戦しておく必要がある。しかし、学習が足りない場合や、事前の対戦では出なかった盤面になった場合には使えないという欠点もある。

2.2 オセロの一般的な戦略での移行判断基準

オセロの一般的な戦略としてあげられるもので、序盤、中盤、終盤にわけられる。その判断基準としては次の 2 通りがあり、一定の手数の経過、特定の場所に石が置かれた場合である。

2.2.1 一定の手数の経過

オセロゲームには、ほかの盤ゲームとは違い、最大 60 手となることが決まっており、一定の手数の経過で序盤、中盤、終盤へと戦略を移行させます。この方法では、盤面の状況とは一切関係なく、手数で判断するので誤った手を打つなどのミスが減る。

2.2.2 特定の場所に石が置かれた場合

この方法は手数などの判断基準と違ってある場所に石を置かれたら序盤から中盤へと戦略を移行するなど置かれた場所によって戦略を変えていく方法である。ただしこの戦略は序盤から終盤への移行が早い場合に終盤の戦略を行う判断基準が低くなるという問題がある。

2.3 先読みと評価関数

定石データベースでは序盤のみ、完全読みは残り手が 25～30 以下での終盤のみ使用することができる。そこで、一般的には評価関数を用いて現在の盤面、または数手先

の盤面を評価する。評価関数として、一般的に誰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 8×8 マスの全てに重みをつけてある局面をその値で評価する。ただ、盤面評価値で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った問題がある。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 個に分けられる。そのため先読み局面評価を行うためには、まず現局面評価を行える必要があり、まず現局面評価について述べる。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、先に記述したとおり盤面評価、直線性、確定石、直線性、駒数、候補数等様々なものがある。

3 研究内容

オセロプログラムでは、評価関数としてどのようなパラメタを用いれば良いかは明白ではない。本研究では評価関数のパラメタとして盤面に存在する石の位置から評価する盤位置、ひっくり返される可能性が無い位置に置かれた確定石の数、ある局面で次に打てる手の候補数の 3 つを用いる。

3.1 評価関数

本節では、本研究で用いる評価関数について述べる。

3.1.1 確定石(CS) (confirm stone)

確定石とは一度取ると絶対に相手に取られることのない自石の事を指す。確定石はその後の展開に左右されず最後まで自石として残るため、確定石が多いほど有利と考えられる。本研究では全ての確定石を求めるアルゴリズムの作成が困難なため、4 つの辺における確定石のみを評価した。確定石の評価値 CS は以下の式で与えられる。

$$CS = (\text{自分の確定石の数} - \text{相手の確定石の数}) + \text{rnd} * 33$$

100	-40	20	5	5	20	-40	100
-40	-80	-1	-1	-1	-1	-80	-40
20	-1	5	1	1	5	-1	20
5	-1	1	0	0	1	-1	5
5	-1	1	0	0	1	-1	5
20	-1	5	1	1	5	-1	20
-40	-80	-1	-1	-1	-1	-80	-40
100	-40	20	5	5	20	-40	100

図 7　盤位置の評価

3.1.2 盤位置(BP) (Board postion)

盤位置(以下 BP とする)の評価は、8x8 のマス全てに価値を持たせ、自石が置かれていればその値を加算、相手石が置かれていれば減算しその合計値を盤位置の評価値とする。各マスの価値はあらゆる実践データの統計と分析が必要となるため、オリジナルの評価値の作成はしない事とする。各マスの価値は様々なものが提案されている。本研究では図 ~~1~~7 に示す評価値を用いる。この評価を用いる理由として他のパラメタで設定する値より差が大きくなり過ぎない為用いた。盤位置の評価値 BP は以下の式で与えられる。ただし board(i, j)はマス(i, j)が自石なら 1，相手石なら-1，空マスなら 0 となり、BP(i, j)は各マス目の評価値である。また、以下 rnd は 0 から 1 までの一様乱数とする。

$$\mathrm{BP} = \sum_{i=0}^{7}\sum_{j=0}^{7} \mathrm{BP}(i,j) * \mathrm{board}(i,j) * \mathrm{rnd} * 3$$

3.1.3 候補数(NC) (Number of candidates)

候補数とは、ある局面で次に自分、もしくは相手が着手可能なマスの数を表す。一般的に自分の候補数が多いほどよく、相手の候補数が少ないほどよいとされている。候補数の評価値 CN は以下の式で与えられる。

$$CN = (\text{着手可能な候補数} + \mathrm{rnd}*2)*10$$

3.1.4 評価関数

本研究では、上記の確定石 CS, 盤位置 BP, 候補数 NC の 3 つを評価関数のパラメタとして用いる。本研究で用いる評価関数 f は以下の式で与えられる。ただし WBP,WFS,WCN は各パラメタの重みである。

$$f=BP* W_{BP}+CS*W_{CS}+NC*W_{NC}$$

各パラメタに付加する重みの範囲は以下とする。

$$0 \leqq W_{BP} \leqq 5 \qquad 0, \leqq W_{FS} \leqq 5 \qquad 0, \leqq W_{CN} \leqq 1$$

3.2 オセロプログラムでの実験

本研究では、評価関数の各要素のどれがよりも正確に各局面の評価値を反映させているかを検討するために、各戦略変化させ、1000 回計算実験を行った。付録に本研究で作成したオセロプログラムのソースを示す。

本研究で作成した主なプログラムは、以下のクラスから成る。

OSERO クラス

OSERO クラスは Int board(),int ConfirmStone(), int NumberCandidates()の、これらで関数を用いて評価値を計算する。

Int board()は、盤面の情報を記憶し、着手可能なマスを記憶させ、着手可能なら true,それ以外を false,とした。

Int ConfirmStone()は局面の 4 辺の確定石を計算し、それを戻り値としてもつ。

また、int [] ConfirmStone2(),int []NumberCandidates2 (),で着手する手の決定、

Int board で、各局面での盤位置を計算し、それを戻り値としてもつ。

Int ConfirmStone で局面の 4 辺の確定石を計算し、それを戻り値としてもつ。

Int NumberCandidates で着手可能なますを計算し、それを戻り値としてもつ。

4 結果と考察

各戦略に対し、先手および後手でランダムオセロプログラムと対戦したときの対戦結果、およびその結果から得られるパラメタの最適な重みについて考察する。

4.1 各パラメタの効力

評価関数のパラメタとして、各パラメタを単独で用いた場合の、対戦結果を表 1.に示す。表 1 より先手後手に関わらずほとんどの場合でパラメタ CS が最も効力を表していることがわかる。表 1. より以下の効力の優先順位が予測できる。

$$CS > BP > NC$$

4.2 CS と BP と NC での比較

CS と BP と NC の三つのパラメタを用いたときの対戦結果を表 2 に示す。表 2 より先手、後手ともに[2:5:1](=[WBP,WFS,WCN)が有効であることが示される。また表 2 から表 4 にかけて先手、後手ともに勝数の上位 3 位までを抽出しその重みの分布を示す。

表 1 各パラメタでの勝率

	先手			後手		
	勝	負	分	勝	負	分
BP	739	220	41	741	210	49
CS	841	133	32	823	151	26
NS	651	329	20	612	368	20
BP*1+CS*3	987	9	4	983	10	7
BP*5+NC*1	807	152	47	806	165	29
CS*5+NC*1	816	165	23	847	133	20
BP*2+CS*5+NC*1	979	14	7	971	21	8

表 2. F=BP*WBP+CS*WCS+NC*WNC の対戦結果 1

BP	CS	NC	先手			後手		
			勝	負	引	勝	負	引
1	1	1	907	68	25	928	55	17
		2	873	106	21	865	113	22
		3	822	144	34	832	140	28
		4	816	157	27	807	165	27
		5	764	203	33	755	214	31
	2	1	946	35	19	958	34	8
		2	931	59	10	892	93	15
		3	892	93	15	887	90	23
		4	847	132	21	854	123	23
		5	829	150	21	825	153	20
	3	1	961	28	11	967	25	8
		2	921	60	23	946	44	10
		3	895	80	25	915	75	10
		4	887	96	17	896	90	14
		5	881	95	24	862	113	25
	4	1	960	29	11	975	20	5
		2	937	41	22	950	43	7
		3	940	46	14	924	61	15
		4	915	65	20	919	65	16
		5	890	87	23	881	99	20
	5	1	987	9	4	979	15	6
		2	943	51	6	956	36	8
		3	946	41	13	928	57	15
		4	894	83	23	922	62	16
		5	913	72	15	924	66	10

表 3 F=BP*WBP+CS*WCS+NC*WNC の対戦結果 2

BP	CS	NC	先手			後手		
			勝	負	引	勝	負	引
2	1	1	848	121	31	846	125	29
		2	868	103	29	855	110	35
		3	839	126	35	806	173	21
		4	821	155	24	809	156	35
		5	818	150	32	796	171	33
	2	1	920	63	17	907	68	25
		2	894	82	24	908	72	20
		3	866	106	28	888	87	29
		4	854	122	24	861	127	12
		5	831	140	33	835	137	28
	3	1	951	40	8	943	47	10
		2	923	61	16	936	54	10
		3	916	71	13	914	65	21
		4	884	92	24	908	80	12
		5	876	104	20	885	95	20
	4	1	953	30	17	966	25	9
		2	954	40	6	955	30	15
		3	949	38	13	931	53	16
		4	923	60	17	921	65	14
		5	906	74	20	915	64	21
	5	1	962	24	14	958	24	18
		2	956	29	15	960	29	11
		3	933	51	16	935	53	13
		4	933	55	12	916	69	15
		5	912	76	12	914	66	20

表 4 F=BP*WBP+CS*WCS+NC*WNC の対戦結果 3

BP	CS	NC	先手			後手		
			勝	負	引	勝	負	引
3	1	1	845	126	29	835	130	35
		2	835	139	26	845	115	35
		3	855	125	20	827	138	35
		4	789	167	44	801	168	31
		5	816	155	29	826	136	38
	2	1	896	83	21	913	63	24
		2	900	87	22	905	78	20
		3	887	89	24	876	101	23
		4	862	114	24	870	109	21
		5	848	115	37	840	123	36
	3	1	941	51	10	940	94	16
		2	926	57	17	919	65	16
		3	901	83	18	908	70	22
		4	885	94	21	907	78	15
		5	878	99	23	876	98	26
	4	1	943	40	17	945	35	20
		2	933	51	16	935	54	11
		3	918	59	23	926	53	21
		4	922	64	14	908	73	19
		5	898	87	15	907	82	13
	5	1	961	29	10	970	24	6
		2	945	34	21	943	42	15
		3	940	46	14	937	43	20
		4	918	68	12	926	59	15
		5	922	64	14	913	74	13

5 結論・今後の課題

本研究では、盤面の評価関数を使ったオセロプログラムを作ろうとしたが、盤面の評価値だけを使ったオセロプログラムでは強い AI はあまりできない。そこで、オセロの局面の評価値の最適な評価関数を求めるために評価関数の各パラメタの重みを変化させながら実験した。本研究により局面の評価値を求める評価関数が 3 つのパラメタ確定石 CS,盤位置 BP,候補数 NC を持つとき各パラメタには、CS>BP>NC となった。
また、本研究で盤面の評価値だけでは強いプログラムはできなかったため、確定石と盤位置と候補数の評価関数も使い実験を行った。

本研究では最初の局面で盤面評価値をだし各局面では評価関数をつかい実験をおこなったが、局面ごとに盤面評価値と評価関数を使うとより最適な結果がでるかどうかを今後の課題としたい。

謝辞

本研究を行うにあたり、直接指導して頂いた近畿大学理工学部情報学科情報論理工学研究室石水講師には大変お世話になり、日頃の研究に関する議論や研究のサポート、研究へのアドバイス、論文指導に対し適切なご助言と励ましなどを頂きましたので、ここに感謝の意を表します。

参考文献

[1] Seal software,リバーシのアルゴリズム C++&Java 対応,工学社(2003)

[2] Koso Sato, 評価関数を考える, プログラミングティーショップ(2003) http://www.geocities.co.jp/SiliconValley-Bay/4543/Osero/Value/Value.html.

[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] 大筆豊, オセロプログラムの評価関数の改善について, 情報処理学会研究報告 2004-G1-11, pp.15-20(2004) http://id.nii.ac.jp/1001/00058554/

[5] リバーシ/オセロ,http://www.mix-zone.net/

[6] 最強プログラム HAYABUSA , http://camp.tablegames.jp/othello_hayabusa_lp.php

[7] Yuichi, Sundry Street(2012), http://www2u.biglobe.ne.jp/~yuichi/

[8] Joel Feinstein, Amenor Wins World 6x6 Championships!, Forty billion noted under the tree (July 1993), pp.6-8,British Othello Federation's newsletter., (1993), http://www.britishothello.org.uk/fbnall.pdf

[9] 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).

[10] Richard Delrme, Ohello programing, (2012), http://abulmo.perso.neuf.fr/index.htm

[11] 石井隆, Master Reversi, (2011), http://homepage2.nifty.com/t_ishii/mr/index.html

[12] Gunnar Andersson, WZebra, (2006), http://radagast.se/othello/

[13] 橋本剛, 上田徹, 橋本隼一, オセロ求解へ向けた取り組み, 組合せゲーム・パズル ミニプロジェクト, 第 3 回ミニ研究会, (2008), http://www.lab2.kuis.kyoto-u.ac.jp/~itohiro/Games/Game080307.html#anchor

[14] 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).

[15] Tetsuya Nakajima, Othello!JAPAN(2013), http://www.othello.org/

[16] 作田 誠,人工知能 コンピュータゲームの実装:リバーシ(2012), http://www.ci.sys.fit.ac.jp/

[17] 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

[18] 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

[19] 清慎一, 川嶋俊：探索プログラムによる四路盤囲碁の解, 情報処理学会研究報告, GI 2000(98), pp.69--76 (2000), http://id.nii.ac.jp/1001/00058633/

[20] 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).

[21] Michael Buro,Logistello,https://skatgame.net/mburo/log.html,1997

[22] Richard Delrme, Ohello programing, 2012, http://abulmo.perso.neuf.fr/index.htm

[23] T.Ishii,MasterReversi,他アプリとの対局結果
http://homepage2.nifty.com/t_ishii/mr/gameresult.html,200

付録

以下に本研究で作成したオセロプログラムのソースを示す。

```
public class OSERO{
        private final static int s = 1; //
        private final int N = 3; // N
        private int[][][] bak = new int[99999][N + 2][N + 2];
        private boolean[][] board = new boolean[N + 2][N + 2];
        private int[][] pList;
                  int pSize;
                 final static int empty = 0;
       public final static int white = 1;
           final static int black = -1;
           final static int wall = 2;
                      int turn = 1;
                       int time = 0;
/*
(盤位置の評価)
*/
       private final int[][] valueMap =
              {{ 100, -40, 20, 5, 5, 20, -40, 100 },
              { -40, -80, -1, -1, -1, -1, -80, -40},
              { 20, -1, 5, 1, 1, 5, -1, 20 },
              { 5, -1, 1, 0, 0, 1, -1, 5 },
              { 5, -1, 1, 0, 0, 1, -1, 5 },
              { 20, -1, 5, 1, 1, 5, -1, 20 },
              { -40, -80, -1, -1, -1, -1, -80, -40 },
              { 100 -40, 20, 5, 5, 20, -40, 100 } };
              /*
              Boardをリセット
              /*
public void resetBoard() {
    for (int y = 0; y < N + 2; y++) {
    for (int x = 0; x < N + 2; x++) {
          if (x == 0) {
             board[y][x] = wall;
```

```
                    } else if (x == N + 1) {
                      board[y][x] = wall;
} else if (y == 0) {
                    board[y][x] = wall;
                                } else if (y == N + 1) {
                    board[y][x] = wall;
                    board[y][x] = empty;
                }
        }
   }
                    board[N / 2][N / 2] = white;
                    board[N / 2 + 1][N / 2 + 1] = white;
                    board[N / 2 + 1][N / 2] = black;
                    board[N / 2][N / 2 + 1] = black;
/*
配置可能かを判断
/*

public boolean isPossible(int x, int y) {
           setPossibleBoard();
                    return pboard[y][x];
                  }
/*
配置可能なマスが存在するか探す
/*
public boolean isPossible() {
      setPossibleBoard();
         for (int y = 0; y < N + 2; y++) {
         for (int x = 0; x < N + 2; x++) {
    if (pboard[y][x]) {
         }
 }
    return false;
       }
       /*
       サイズを求める
```

```
        /*
public void setPSize() {
     pSize = 0;
      for (int y = 0; y < N + 2; y++) {
       if (pboard[y][x]) {
            pSize += 1;
  }
      }
             }
      }
      /*
      Stets の取得
      /*
public int getstets() {
      return stets;
}
/*
初期の盤目から配置可能なマスを探す
/*
private void Board() {
for (int y = 0; y < N + 2; y++) {
     for (int x = 0; x < N + 2; x++) {
         board[y][x] = false;
}
      }
int piece = gebb();
int cPiece = gebb() * (-1);
      for (int y = 0; y < N + 2; y++) {
       for (int x = 0; x < N + 2; x++) {
if (board[y][x] == iece) {
   sX = x;
    sY = y;
if (board[sY - 1][sX - 1] == iece) {
               do {
          sX--;
        sY--;
```

```
      } while (board[sY][sX] == iece)

if (board[sY][sX] == empty) {
   pboard[sY][sX] = true;
      }
 }
 sX = x;
 sY = y;
if (board[sY - 1][sX] == iece) {
         do {
         sY--;
} while (board[sY][sX] == iece);
     if (board[sY][sX] == empty) {
        pboard[sY][sX] = true;
      }
 }
   sX = x;
    sY = y;
if (board[sY - 1][sX + 1] == iece) {
         do {
         sX++;
         sY = y;
if (board[sY - 1][sX + 1] == iece) {
     do {
        sX++;
        sY--;
      } while (board[sY][sX] == iece);
if (board[sY][sX] == empty) {
        pboard[sY][sX] = true;
          }
      }

      sX = x;
      sY = y;
 if (board[sY][sX - 1] == iece) {
     do {
```

```
      sX--;
        } while (board[sY][sX] == iece);
       if (board[sY][sX] == empty) {
       pboard[sY][sX] = true;
       }
      }

        sX = x;
        sY = y;
if (board[sY][sX + 1] == iece) {

        sX++;
} while (board[sY][sX] == iece);
    if (board[sY][sX] == empty) {
      pboard[sY][sX] = true;
      }
      }
        sX = x;
        sY = y;
if (board[sY + 1][sX - 1] == iece) {
              do {
              sX--;
              sY++;
} while (board[sY][sX] == iece);
       if (board[sY][sX] == empty) {
       pboard[sY][sX] = true;
                sX = x;
                sY = y;
      if (board[sY + 1][sX] == iece) {
       do {
                sY++;
       } while (board[sY][sX] == iece);
          if (board[sY][sX] == empty) {
            pboard[sY][sX] = true;
                    }
          }
```

```
                sX = x;
                sY = y;
    if (board[sY + 1][sX + 1] == iece) {
    do {
                sX++;
                sY++;
    } while (board[sY][sX] == iece);
    if (board[sY][sX] == empty) {
      pboard[sY][sX] = true;
       }
              }
                  }
                                                  }
                        }
              }

public int getbb() {
     return turn;
     }
/*
盤面の配置可能マスを加えて表示
/*
public void Board() {
       for (int i = 1; i <= N; i++) {
          System.out.print(" " + i);
     }
          System.out.println();
             for (int y = 1; y < N + 1; y++) {
          System.out.print(y);
             for (int x = 1; x < N + 1; x++) {
     if (board[y][x] == white) {
             System.out.print("○");
                              } else if (board[y][x] == black) {
             System.out.print("●");
                                   } else {
             System.out.print("□");
```

```
                                    }
                            }
                System.out.println();
                                }
                        }
/*
盤面の表示
/*
public void Board() {

                        for (int i = 1; i <= N; i++) {
        System.out.print(" " + i);
    }
        System.out.println();
                        for (int y = 1; y < N + 1; y++) {
        /*
        マスの反転
        /*
public void reversiPiece(int[] input) {

                int ix = input[0];
                int iy = input[1];
    board[iy][ix] = getbb();
                  int piece = getbb();
        int cPiece = getbb() * (-1);
        int sx, sy;
                        if (board[iy - 1][ix - 1] == ece) {
                                        sx = ix - 1;
                                        sy = iy - 1;
                                            do {
                                    sx--;
                                    sy--;
} while (board[sy][sx] == ece);
                        if (board[sy][sx] == piece) {
                                    sx = ix - 1;
                                    sy = iy - 1;
```

```
                                    do {
                        board[sy][sx] = piece;
                                    sx--;
                                     sy--;
} while (board[sy][sx] == iece);
      }
                              if (board[iy - 1][ix] == iece) {
                                        sx = ix;
                                        sy = iy - 1;
                                                do {
                                         sy--;
} while (board[sy][sx] == ece);
                                 if (board[sy][sx] == ece) {
                                        sx = ix;
                                        sy = iy - 1;
                                           do {
board[sy][sx] = piece;
             sy--;
} while (board[sy][sx] == iece);
          }
}


if (board[iy - 1][ix + 1] == iece) {
            sx = ix + 1;
         sy = iy - 1;
             do {
         sx++;
          sy--;
} while (board[sy][sx] == iece);
                           if (board[sy][sx] == iece) {
                                    sx = ix + 1;
                                    sy = iy - 1;
do {
board[sy][sx] = iece;
            sx++;
            sy--;
```

```
} while (board[sy][sx] == iece);
          )    }
                    if (board[iy][ix - 1] == iece) {
                              sx = ix - 1;
                              sy= ix + 1;
                              sy = iy;
                                  do {
                                        sx++;
} while (board[sy] [sx] == iece);
                   if (board[sy][sx] == piece) {
                              sx = ix + 1;
                              sy = iy;
do {
board[sy][sx] = piece;
            sx++;
} while (board[sy][sx] == iece);
        )
 }
                if (board[iy + 1][ix - 1] == iece) {
                              sx = ix - 1;
                              sy = iy + 1;
                                       do {
                              sx--;
                              sy++;
} while (board[sy][sx] == iece);
                if (board[sy][sx] == piece) {
                              sx = ix - 1;
                              sy = iy + 1
               do {
board[sy][sx] = ece
                sx--;
                sy++;
} while (board[sy][sx] == iece);
                                                              }
                                                    }
```

```
                    if (board[iy + 1][ix] == iece) {
                        sx = ix;
                        sy = iy + 1;
                        do {
                    sy++;
} while (board[sy][sx] == iece);
                    if (board[sy][sx] == piece) {
                       sx = ix;
                         sy = iy + 1;
            do {
           board[sy][sx] = piece;
                    sy++;
                } while (board[sy][sx] == iece);
              if (board[sy][sx] == piece) {
                        sx = ix;
                        sy = iy + 1;
     do {
   board[sy][sx] = piece;
                    sy++;
            } while (board[sy][sx] == cPiece);
                }
       }
                        if (board[iy + 1][ix + 1] == cPiece) {
                            sx = ix + 1;
                  sy = iy + 1;
                  do {
                    sx++;
                     sy++;
} while (board[sy][sx] == cPiece);
            if (board[sy][sx] == piece) {
                sx = ix + 1;
                 sy = iy + 1;
       do {
            board[sy][sx] = piece;
                        sx++;
                        sy++;
```

```
                } while (board[sy][sx] == cPiece
/*
ターン交代
/*
public void turnChange() {
                    turn *= (-1);
                        }

public void consectiveTurnChange() {
                    time += 1;
                    turn *= (-1);
                      }
/*
タイムリセット
/*
public void timeReset() {
                time = 0;
                   }
/*
時間の表示
/*
public boolean timeWatcher() {
                      if (time >= 2) {
                      return false;
                                          } else {
        return true;
                                      }
                       }
/*
白駒の計算
/*
public int countWhite() {
              int count = 0;
                  for (int y = 0; y < N + 2; y++) {
                  for (int x = 0; x < N + 2; x++) {
if (board[y][x] == white) {
```

```
                        count += 1;
                            }
                            }
            }
                return count;
                    }
/*
黒駒の計算
/*
public int countBlack() {
                    int count = 0;
                        for (int y = 0; y < N + 2; y++) {
                        for (int x = 0; x < N + 2; x++) {
if (board[y][x] == black) {
                                count += 1;
                                        }
                            }
                        }
return count;
                }
public void setPList() {
                        setPossibleBoard();
                            setPSize();
pList = new int[pSize][2];
                            int i = 0;
                            for (int y = 0; y < N + 2; y++) {
                            for (int x = 0; x < N + 2; x++) {
if (pboard[y][x]) {
                    pList[i][0] = x;
                    pList[i][1] = y;
                            i++;
                                }
                    }
                }
    }
    /*
```

プレイヤーの表示

```
/*
public int[] player() {
                        int[] input = new int[2];
                         int inputX;
                         int inputY;
Scanner kbs = new Scanner(System.in);
                do {
System.out.print("x:");
                        inputX = kbs.nextInt();
                          if (inputX <= 0 || inputX >= 9) {
System.out.println("x は1-8 で入力してください");
                          }
              }
while (inputX <= 0 || inputX >= 9);
                  do {
System.out.print("y:");
                    inputY = kbs.nextInt();
                    if (inputY <= 0 || inputY >= 9) {
System.out.println("y は1-8 で入力してください");
                     }
                          } while (inputY <= 0 || inputY >= 9);
                                input[0] = inputX;
                                input[1] = inputY;
                    return input;
                              }
                     }
int MaxValue[] = { -999999, -1 };
                    int value;
                        for (int i = 0; i < pSize; i++) {
                         value = evaluateMap(pList[i]);
                              if (MaxValue[0] < value) {
                            MaxValue[0] = value;
                            MaxValue[1] = i;
                         }
              }
```

```
return pList[MaxValue[1]];
                        }
/*
確定石が0の時評価値の高い座標を返す
/*
public int[] valueConfirmComputer() {
                          setPList();
                        int MaxValue[] = { -999999, -1 };
                          int value;
                          int zero = 0;
for (int i = 0; i < pSize; i++) {
                         value = evaluateFinal(pList[i]);
                              zero += value;
if (MaxValue[0] < value) {
                                MaxValue[0] = value;
                                MaxValue[1] = i;
                              }
                    }
if (zero == 0) {
return pList[(int) Math.floor(Math.random() * (pSize))];
                                }
return pList[MaxValue[1]];
                                }
public int[] valueCNComputer() {
                                setPList();
              int MaxValue[] = { -999999, -1 };
                  int value;
                                for (int i = 0; i < pSize; i++) {
value = evaluateCN(pList[i]);
                        if (MaxValue[0] < value) {
                              MaxValue[0] = value;
                              MaxValue[1] = i;
                                 }
                  }
return pList[MaxValue[1]];
                        }
```

```
/*
盤面評価と確定石
/*
public int[] valueMapFinalComputer(int a, int b) {
                        setPList();
                          int MaxValue[] = { -999999, -1 };
                           int value;
                      for (int i = 0; i < pSize; i++) {
                  value = evaluateMapFinal(pList[i], a, b);

if (MaxValue[0] < value) {
                           MaxValue[0] = value;
                           MaxValue[1] = i;
                                                    }
                                                }
                           return pList[MaxValue[1]];
                                                }
/*
盤面評価と候補数
/*
public int[] valueMapNumberCondidatecomputer(int a, int b) {
                    setPList();
                        int MaxValue[] = { -999999, -1 };
                         int value;
                            for (int i = 0; i < pSize; i++) {
value = evaluateMapCN(pList[i], a, b);
                           if (MaxValue[0] < value) {
                              MaxValue[0] = value;
                              MaxValue[1] = i;
                           }
                }
return pList[MaxValue[1]];
                        }
/*
確定石と候補数 評価値の高い座標を返す
/*
```

```
public int[] valueConfirmNumberCondidateComputer(int a, int b) {
                                setPList();
                                int MaxValue[] = { -999999, -1 };
                                int value;
                                   for (int i = 0; i < pSize; i++) {

value = evaluateFinalCN(pList[i], a, b);
                               if (MaxValue[0] < value) {
                                  MaxValue[0] = value;
                                  MaxValue[1] = i;
                                   }
                        }
return pList[MaxValue[1]];
                              }
/*
盤面評価と確定石の評価値の高い方を返す
/*
public int[] valueMapConfirmNumberCondidateComputer(int a,int b,int c) {
                           setPList();
                                int MaxValue[] = { -999999, -1 };
                                int value;
                          for (int i = 0; i < pSize; i++) {
                              value = evaluateMapFinalCN(pList[i],a,b,c);
if (MaxValue[0] < value) {
                              MaxValue[0] = value;
                              MaxValue[1] = i;
                                          }
                         }
return pList[MaxValue[1]];
                               }
/*
引数で与えた座標に置いたときの評価値をあらわす
/*
public int evalueateboard() {

                                    int value = 0;
```

```
                                    for (int y = 1; y < N + 1; y++) {
                                    for (int x = 1; x < N + 1; x++) {
                        if (board[y][x] != 0) {
value += (board[y][x] * valueMap[y - 1][x - 1]) * getTurn();
                                        }
                                    }
                    }
                    return value;
                            }
/*
引数で与えた座標に置いたときの確定石
/*
public int evaluateMap(int[] piece) {
                            int value = 0;
                            int[][] boardbak = new int[N + 2][N + 2];
                            int myTurn = getTurn();
copyBoard(board, boardbak);
                        reversiPiece(piece);
                                for (int y = 1; y < N + 1; y++) {
                                for (int x = 1; x < N + 1; x++) {
if (board[y][x] != 0) {
                        value += (board[y][x] * (valueMap[y - 1][x - 1]
                       + (int) Math.floor(Math.random() * 3)));
                                }
                    }
            }
if (myTurn == -1) {
                        value *= -1;
                                }
copyBoard(boardbak, board);
                            // System.out.printf("(%d , %d ) %d \n", piece[0], piece[1], value);
                                    return value;
                                }

public int evaluateFinal(int[] piece) {
                                int value = 0;
```

```
                                int[][] boardbak = new int[N + 2][N + 2];
                                 copyBoard(board, boardbak);
                        reversiPiece(piece);
                                value = evaluateFinalStone();
                                 copyBoard(boardbak, board);
// System.out.printf("(%d , %d ) %d \n", piece[0], piece[1], value);
                            return value;
                                }
/*
引数で与えた座標の盤面評価と確定石をあらわす
/*
public int evaluateMapFinal(int[] piece, int a, int b) {
                                    int value = 0;
                                    int[][] boardbak = new int[N + 2][N + 2];
                                    int myTurn = getTurn();
            int FS = 0, BP = 0;
                        copyBoard(board, boardbak);
                        reversiPiece(piece);
                                     for (int y = 1; y < N + 1; y++) {
                                     for (int x = 1; x < N + 1; x++) {
                                     if (board[y][x] != 0) {
                    BP += (board[y][x] * (valueMap[y - 1][x - 1]
                + (int) Math.floor(Math.random() * 3)));
                                        }
                                   }
                    }
if (myTurn == -1) {
                                BP *= -1;
                     }
FS = evaluateFinalStone();
                                 copyBoard(boardbak, board);
                                   value = (a * BP) + (b * FS);
                                  // System.out.printf("(%d , %d ) %d \n", piece[0], piece[1], value);
return value;
                                        }
*/
```

評価値の計算 候補数

```
/*
public int evaluateMapNUmberConfirm(int[] piece, int a, int b) {
                                        int CN = 0;
                                int[][] boardbak = new int[N + 2][N + 2];
                                int myTurn = getTurn();
                                int value = 0, BP = 0;
// board のバックアップ作成
                                        copyBoard(board, boardbak);
                                       reversiPiece(piece);
                                for (int y = 1; y < N + 1; y++) {
                                for (int x = 1; x < N + 1; x++) {
if (board[y][x] != 0) {
                                 BP += (board[y][x] * (valueMap[y - 1][x - 1]
                                 + (int) Math.floor(Math.random() * 3)));
                                        }
                                }
                        }
if (myTurn == -1) {
                                BP *= -1;
                                 }
CN = pSize + (int) Math.floor(Math.random() * 2);
                        // board の復元
                                        copyBoard(boardbak, board);
                                       value = (BP * a) + (CN * 10 * b);


                                                                return value;
                                          }
/*
評価値の計算 確定石と候補数
/*
public int evaluateFinalNumberConfirm(int[] piece, int a, int b) {
                                       int[][] boardbak = new int[N + 2][N + 2];
                                          int value = 0;
                                          int FS = 0, CN = 0;
                                copyBoard(board, boardbak);
```

```
                                reversiPiece(piece);
                                                FS = evaluateFinalStone();
                                               CN = pSize + (int) Math.floor(Math.random() * 2);
                            copyBoard(boardbak, board);
                                 turnChange();
                                 setPSize();
                                value = (FS * 1) + (CN * 10 * b);
                                return value;
/*
評価値を計算しそれを返す 盤面評価 確定石 候補数
/*
public int evaluateMapFinalCN(int[] piece,int a,int b,int c) {
                                int[][] boardbak = new int[N + 2][N + 2];
                                int myTurn = getTurn();
                                int value = 0;
                                 int BP=0,FS=0,CN=0;
// board のバックアップ作成
                           copyBoard(board, boardbak);
                              reversiPiece(piece);
                                      for (int y = 1; y < N + 1; y++) {
                                      for (int x = 1; x < N + 1; x++) {
               if (board[y][x] != 0) {
                                  BP += ((board[y][x] * valueMap[y - 1][x - 1])
                                  +(int) Math.floor(Math.random() * 3));
                                          }
                                 }
                 }
if (myTurn == -1) {
                              BP *= -1;
                         }
CN = pSize + (int) Math.floor(Math.random() * 2);
                                copyBoard(boardbak, board);
                                turnChange();
                                setPossibleBoard();
                               setPSize();
value = (a*BP)+(b*FS)+(CN*10*c);
```

```
                return value;
/*
引数で与えた座標においた時の評価値を返す
/*
public int evaluateNumberConfirm(int[] piece) {
        int NC = 0;
        int[][] boardbak = new int[N + 2][N + 2];
                                        copyBoard(board, boardbak);
reversiPiece(piece);
                        turnChange();
                                setPossibleBoard();
                                setPSize();
            CN = pSize;
                  copyBoard(boardbak, board);
                 turnChange();
               // ppList,ppSize の復元
               setPossibleBoard();
setPSize();
return -((CN+(int) Math.floor(Math.random() * 2)) * 10);
               }
/*
盤面のコピー
/*
public void copyBoard(int[][] a, int[][] b) {
                for (int y = 0; y < N + 2; y++) {
                for (int x = 0; x < N + 2; x++) {
                                b[y][x] = a[y][x];
                                    }
                        }
        }
public int getNullBoard() {
                 int num = 0;
                        for (int y = 1; y < N + 2; y++) {
                        for (int x = 1; x < N + 2; x++) {
if (board[y][x] == 0) {
                num++;
```

```
                        }
                    }
                }
        return num;
}
/*
  minlevel とmaxlevel でミニマックス法を実施
/*
public int[] search_computer(int s) {
                        int[] answer = new int[2];
                        int value, value_max = -999999;
                                setPList();
            int[][] list = new int[pSize][2];
                        for (int y = 0; y < pSize; y++) {
                        for (int x = 0; x < 2; x++) {
                                list[y][x] = pList[y][x];
                                    }
                     }
int size = pSize;
                       int nullNum = getNullBoard();
                        if (s > nullNum) {
                        s = nullNum;
                       }
System.out.println(nullNum);
                    for (int i = 0; i < size; i++) {
                     copyBoard(board, bak[s]);
                     reversiPiece(list[i]);
                      turnChange();
                            value = minlevel(s - 1);
System.out.println("x:" + list[i][0] + ",y:" + list[i][1]
                            + ",value:" + value);
                       turnChange();
                        copyBoard(bak[s], board);
if (value > value_max) {
                        answer[0] = list[i][0];
                        answer[1] = list[i][1];
```

```
                    }
                }
return answer;
            }

public int maxlevel(int limit) {
                    if (limit == 0) {
                    return evalueateboard();
                }
/*
配置可能な手を生成
/*
                setPList();
int[][] list = new int[pSize][2];
                for (int y = 0; y < pSize; y++) {
                for (int x = 0; x < 2; x++) {
list[y][x] = pList[y][x];
                    }
        }
int score, score_max = -99999;
                    for (int i = 0; i < list.length; i++) {
copyBoard(board, bak[limit]); reversiPiece(list[i]);
                turnChange();
                    score = minlevel(limit - 1);
                    turnChange();
copyBoard(bak[limit], board); if (score > score_max) {
                    score_max = score;
                        }
                    }
return score_max;
                }
public int minlevel(int limit) {
                    if (limit == 0) {
return evalueateboard();
                }
/*
```

配置可能な手を生成

```
/*
setPList();
                        int[][] list = new int[pSize][2];
                                for (int y = 0; y < pSize; y++) {
                                for (int x = 0; x < 2; x++) {
list[y][x] = pList[y][x];
                                 }
                    }
int score, score_min = 99999;
                         for (int i = 0; i < list.length; i++) {
                                    copyBoard(board, bak[limit]);/ reversiPiece(list[i]);
                                    turnChange();
                                    score = maxlevel(limit - 1);
                                    turnChange();
                                    copyBoard(bak[limit], board); if (score < score_min) {
                                    score_min = score;
                                                }
                                }
                                                        return score_min;
                            }
public int[] randomComputer() {
                              setPList();
                                        return pList[(int) Math.floor(Math.random() * (pSize))];
                               }
public int evaluateFinalStone() {
                                 int valueOfFinalStone = 0;
                                 int myTurn = getTurn();
                                    int i;
                                    boolean full;
                                        int stonePoint = 11;
                                            int HL = board[1][1], HR = board[1][N], /
   LL = board[N][1], /
   LR = board[N][N];/
if (HL != empty || HR != empty || LL != empty || LR != empty) {
/*
```

マスが埋まっているかを判断

```
/*
                full = true;
                    for (int j = 0; j < N + 1; j++) {
/*
empty があればfalse
/*
                    if (board[1][j] == empty) {
                     full = false;
                                }
                      }
/*
全て埋まっている
/*
if (full) {for (int j = 1; j < N + 1; j++) {
                valueOfFinalStone += ((board[1][j] * myTurn * stonePoint)
                        + (int) Math.floor(Math.random() * 3));
                  }
                    if (HL != empty) {
                         i = 1;
while (board[1][i] == HL) {
                         valueOfFinalStone += ((board[1][i] * myTurn * stonePoint)
                          + (int) Math.floor(Math.random() * 3));
                                  i++;
                          }
            }
                    i = N;
while (board[1][i] == HR) {
                         valueOfFinalStone += ((board[1][i] * myTurn * stonePoint)
                                 + (int) Math.floor(Math.random() * 3));
                         i--;
                                      }
                                }
                    }
                        full = true;
```

```
                for (int j = 1; j < N + 1; j++) {
// empty があればfalse
                    if (board[N][j] == empty) {
                    full = false;
                        }
                    }
            if (full) {/ */
                for (int j = 1; j < N + 1; j++) {
                valueOfFinalStone += ((board[N][j] * myTurn * stonePoint)
                + (int) Math.floor(Math.random() * 3));
             }
                i = 1;
while (board[N][i] == LL) {
                    valueOfFinalStone += ((board[N][i] * myTurn * stonePoint)
                    + (int) Math.floor(Math.random() * 3));
                        i++;
                        }
                    }

                if (LR != empty) {
                  i = N;
while (board[N][i] == LR) {
                    valueOfFinalStone += ((board[N][i] * myTurn * stonePoint)
                     + (int) Math.floor(Math.random() * 3));
                            i--;
                              }
                     }
                    }
                full = true;
                for (int j = 1; j < N + 1; j++) {
                    if (board[j][1] == empty) {
                    full = false;
                            }
                 }
if (full) {/* for (int j = 1; j < N + 1; j++) {
                    valueOfFinalStone += ((board[j][1] * myTurn * stonePoint)
```

```
                        + (int) Math.floor(Math.random() * 3));
                        }
                    }
if (HL != empty) {
                     i = 1;
while (board[i][1] == HL) {
                        valueOfFinalStone += ((board[i][1] * myTurn * stonePoint)
                        + (int) Math.floor(Math.random() * 3));
                            i++;
                    }
              }
                       i = N;
while (board[i][1] == LL) {
                        valueOfFinalStone += ((board[i][1] * myTurn * stonePoint)
                        + (int) Math.floor(Math.random() * 3));
                        i--;
                }
          }
  }
                        full = true;
                        for (int j = 1; j < N + 1; j++) {
// empty があればfalse
                        if (board[j][N] == empty) {
                        full = false;
                                }
                    }
              if (full) for (int j = 1; j < N + 1; j++) {
            valueOfFinalStone += ((board[j][N] * myTurn * stonePoint)
             + (int) Math.floor(Math.random() * 3));
                        }

                if (HR != empty) {
                i = 1;
while (board[i][N] == HR) {

                  valueOfFinalStone += ((board[i][N] * myTurn * stonePoint)
```

```
                    + (int) Math.floor(Math.random() * 3));
                        i++;
                            }
                    }
// 右下は埋まっている
                        if (LR != empty) {
                    i = N;
while (board[i][1] == LR) {
                            valueOfFinalStone += ((board[i][N] * myTurn * stonePoint)
                        + (int) Math.floor(Math.random() * 3));
                            i--;
                            }
                        }
                    }
  }
valueOfFinalStone -= ((HL + HR + LL + LR) * 10);
                        return valueOfFinalStone;
            }
/*
* randomComputer ランダム valueMapComputer 盤面評価 valueFinalComputer 確□□□□□□□□□□定石
* valueCNComputer 候補数 valueMapFinalComputer 盤面評価と確定石 valueMapCNComputer
* 盤面評価と候補数 valueFinalCNComuter 確定石と候補数
*/
public static void main(String[] args) {
                        Date before = new Date();
                            Reversi board = new Reversi();
int input[];
                    for (int a = 1; a < 6; a++) {
                    for (int b = 1; b < 6; b++) {
/*
                for (int c = 1; c < 6; c++) {
*/
int Win = 0, False = 0, Drow = 0;

                    // System.out.printf("Value:%s,Random:%s\n",
                    // (board.getTurn() == white) ? "○" : "●",
```

```
                    // (board.getTurn() == white) ? "●" : "○");
                     // board.printBoard();
                    for (int i = 0; i < 1000; i++) {
                    board.resetBoard();
                      do {
while (board.isPossible()) {
                     board.timeReset();
                     // System.out.printf("%s のターン\n",
                      // (board.getTurn() == white) ? "Value(○)"
                      // : "Random(●)");
                      // board.printBoardPlusP();
                       do {
// 先手
                     // ValueComputer turn
                     if (board.getTurn() == white) {
                         // input =
                     // board.valueMapFinalComputer();
                     input
                   = board.randomComputer();
                   // RandomComputer turn
                 } else {
// 後手
               input
                = board.valueMapCNComputer
            (a, b);
                  // System.out.printf
                        // ("x:%d, y:%d\n",
                         // input[0],
                         // input[1]);
                 }
                          if(!board.isPossible(input[0], input[1])) {
                         System.out.println
                 ("その座標は打てません.");
                             }
                    } while (!(board.isPossible(input[0], input[1])));
                             board.reversiPiece(input);
```

```
                                board.turnChange();
                        }
                    // System.out.printf
//("%s は打てる場所がありません。\n",(board.getTurn()
// ==
                                // white) ? "Value(○)" : "Random(●)");
                                 board.consectiveTurnChange();
                                 } while (board.timeWatcher());
                                // System.out.println("GameOver");
                                // board.printBoard();
                                // System.out.printf("Value(%s):%d,Random(%s):%d\n",
                                // (board.getTurn() == white) ? "○" : "●",
                                // board.countWhite(),
                                // (board.getTurn() == white) ? "●" : "○",
                                 // board.countBlack());
if (board.countWhite() > board.countBlack()) {
                    Win += 1;
                // System.out.println("win");
                } else if (board.countWhite() < board.countBlack()) {
                            False += 1;
                  // System.out.println("false");
                                } else {
                                            Drow += 1;
                                                                    }
                                        }
                            System.out.println(/* "["+a+":"+b+"]"\t"+ */False + "\t"
                             + Win + "\t" + Drow);
                                                                }
                                            }

                                    }

                        }
                 }
```