卒業研究報告書

題目

4 人版リバーシ Yonin の解析

指導教員

石水 隆　講師

報告者

10-1-037-0040

埒田 基成

近畿大学理工学部情報学科

平成 24 年 1 月 31 日提出

概要

本論文は通常2人でプレイするリバーシを，4人でプレイするよう拡張したYonin[1]について述べる．通常のリバーシは二人零和有限確定完全情報ゲームであり，Yonin はその内プレイヤー人数以外の要素を引き継いでいる．

二人零和有限確定完全情報ゲームにおいて強いAIを作るためには先読み手数を大きくすればよい．これはプレイヤー人数以外の要素を引き継いでいる Yonin でも同様である．しかし，様々な点を考慮することで限られた先読み手数の中でも比較的強いAIは作成可能である．

そこで，本研究ではYoninのAIを作成することで，通常のリバーシから拡張することによるAIの変更点を考察し，より強いYoninのAIを作るためには何が必要かの追求を目指す．

目次

1 序論

1.1 本研究の背景

Yonin はリバーシを 4 人でプレイするように拡張されたものである．通常のリバーシと違い陣地という概念があり，それによって合法手に制限をかけ，勝敗の決定のときに自色の石を数える範囲を狭めている．

リバーシや囲碁などのボードゲームは二人零和有限確定完全情報ゲームに分類される．零和とはプレイヤーの利得の合計が 0 になること．有限とはプレイヤーの指し手の組み合わせが有限であること．確定とはランダム性がないこと．完全情報ゲームとは各プレイヤーが自分の手番に相手の手も含めて，過去，現在の情報を全て知ることができるゲームのことである．二人零和有限確定完全情報ゲームはその性質上解析しやすいためゲーム理論において様々な研究がなされてきた．

1.2 二人零和有限確定完全情報ゲームについての既知の結果

二人零和有限確定完全情報ゲームは双方最善手を指した場合，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，可能な局面数はリバーシが 10^{28} 通り，チェスが 10^{50} 通り，将棋が 10^{69} 通り，囲碁が 10^{170} 通り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[3]．チェッカーは双方最善手を指すと引き分けとなる[4]．

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．サイズ 6x6 のリバーシでは,双方最善手を打つと 16 対 20 で後手勝ちとなる[5]．囲碁は，サイズ 4x4 では双方最善手を打つと持碁（引き分け）になり[6]，5x5 の囲碁は黒の 25 目勝ちとなる[7]．将棋では，盤サイズ 3x4 に減らし，駒の種類を 4 つに減らしたどうぶつしょうぎ[8]が完全解析されており，双方最善手を指すと 78 手で後手が勝つことが判明している[9]．

局面数の大きいゲームに対しては，完全解析で最善手を求めることは現時点では不可能である．そこで最善ではないがより良い手を求める手法として，局面の評価，一定手数の先読み，定跡データベース，対戦データベースの利用，終盤での完全読みなどが使用される．局面の評価とは，あらかじめ盤面の各マスに評価値マップを設定し，置かれている駒と評価値マップを使用して局面ごとの有利不利を計算する手法である．先読みとは，現在の局面から着手可能な手を打った場合の局面を想定して仮想的にゲームを進めていく手法である．定跡データベース，対戦データベースとは，過去の対戦から特定の局面ではどういった着手が有効であるかをあらかじめデータベースに記録しておくことである．終盤での完全読みとは，残りの手数が少なく，局面が限られる終盤で最終手まで読み，最善手を着手することである．これらの手法を用いることにより，リバーシでは人間が太刀打ちできない強さのリバーシプログラムが多く作られている．代表的なリバーシプログラムとしては WZebra[10]，MasterReversi[11]などがある．

1.3 リバーシのバリエーション

リバーシには様々なバリエーションがある．

リバーシの自然な拡張として，盤面サイズが 6x6 の『ミニオセロ』，10x10 の『グランドオセロ』等がある．グランドオセロの各角から 6 マスを取り除くことにより，八角形状の盤を用いる『88 オセロ』，盤面を円形にすることで端を無くし，終盤でも逆転できるようにした『ニップ』[12]等もある．

多人数で対戦できるリバーシとしては，四面体の石を使用する『みんなでオセロ』[13]などが存在する．同様に多人数で対戦できるリバーシとして Yonin[1]がある．Yonin は通常のリバーシで使う 8x8 盤面と石をそのまま使用して 4 人でプレイできるように拡張さたリバーシである．

これらバリエーションのうち，ミニオセロは前述したように完全解析されており，最善手を打ち合った場合，16 対 20 で後手が勝利する[5]．グランドオセロは盤面が広く，着手できる箇所が多いため解析されいない．88 オセロ，ニップ，みんなでオセロなどは解析は進んでいない．

1.4 多人数ゲーム

3 人以上でプレイするゲームを多人数ゲームと言う．多人数ゲームでは，あるプレイヤーが自分の利得と関係無く，他者の利得にのみ影響する行動を取れる，不決定状態と呼ばれる状態が発生する[14]．例えば，最初に一定の得点を獲得したプレイヤー1 人が勝ちとなり，残り全員が負けとなるゲームの場合，早い段階で負けが確定したプレイヤーは，どのような行動を取っても自分の勝敗には影響しないため，あえて特定のプレイヤーの得になる行動を取ることがある．また、複雑なゲームでは，意図しなくとも他のプレイヤーに利益を与える行動を取ってしまうこともある．このため，多人数ゲームでは，完全情報であっても二人ゲームのようにゲーム木探索を一意に行うことができず，最適手を求めることが不可能な場合が起こり得る．

2 人ゲームの着手は，min-max 法，あるいはその改良である alpha-beta 法を用いれば本質的には最善手を選択することができる．一方多人数ゲームの着手の決定には min-max 法の拡張である maxN 法[16]などが用いられるが，不確定要素が多く，問題によっては最適解を得るのが難しい．

1.5 本研究の目的

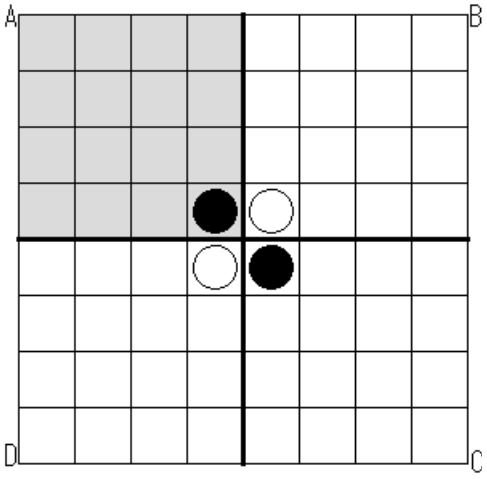
多くの完全情報ゲームでは初手から最終手まで読むことができれば常に最善手を打つことができる．しかし，それには膨大な計算時間が必要になる．このため， 1.2 節で述べたように局面数の大きいゲームに対しては，最善ではないがより良い手を求める手法として，局面の評価，一定手数の先読み，定石データベース，対戦データベースの利用，終盤での完全読みなどが使用される．リバーシに対してはこれらの手法で強いプログラムが存在する．一方で， 1.3 節で述べたリバーシのバリエーションに対しては，マイナーなゲームであるためにあまり解析やプログラムの作成などはされていない．そこで，本研究ではリバーシのバリエーションの 1 つである Yonin に対して，限られた先読み手数の中で探索を行う AI を作成し，通常のリバーシから拡張することによる AI の変更点を考察し，より強い Yonin の AI を作るためには何が必要かの追求を目指す．

1.6 本報告書の構成

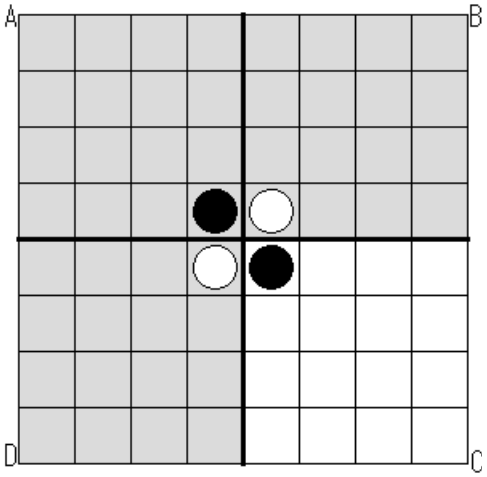
本報告書の構成は以下の通りである．まず第 2 章において，本研究の対象となる Yonin について説明する．続いて第 3 章では本研究で作成した Yonin の AI が打つ手を決定する手法について述べる．第 4 章では作成した AI の性能を見るためにランダムに候補手を打つ AI との対戦結果を記す．第 5 章では第 4 章の結果について考察し，続く第 6 章において結論及び今後の課題を述べる．

2 Yonin について

まず Yonin について簡単に説明する．先に述べたように Yonin とはリバーシを 4 人でプレイできるように拡張されたゲームのことである．使う盤面と石は通常のリバーシと同じものを使い，石の初期配置も通常のリバーシと同じである．この盤面を 4x4 の 4 つの陣地に分け，それぞれのプレイヤーの陣地とする．例として図 1 に，プレイヤーA の陣地を示す．太線で 4 つに分けられた陣地のうち，色部分がプレイヤーA の陣地である．



プレイヤーの石の色は対面のプレイヤー同士が同じになる．着手順は最初のプレイヤーから順に時計回りに着手していく．合法手は基本的には通常のリバーシと同じく相手の色の石を自分の色の石で挟むように置くことができる．ただし，通常のリバーシと違い対面にいるプレイヤーの陣地に石を置くことはできない．例として，図 2 のように，プレイヤーA が石を打てる位置は色部分のみであり，対面にいるプレイヤーC の陣地に石を打つことはできない．



パスやゲームの終了条件も通常のリバーシと同じである．勝敗条件は通常のリバーシとは違い自分の色の石を数えるのは自分の陣地内のみであり，その数の大小で勝敗を決定する．図 1 では，プレイヤーA は色部分の中の自分の色である黒石の数を数える．

3 研究内容

3.1 評価関数

本研究で作成したAI は打てる手が複数ある場合，その手を打った場合に得られる局面を先読みし，それで得られた局面の評価値を用いて手を決定する．

局面を評価する代表的な手法として評価マップの使用がある．評価マップとは盤面の各マスに価値を設定し，マスに自分の石が置かれている場合は評価に価値を足し，相手の石がおかれている場合は評価から価値を引く，というものである．通常のリバーシでは角のマスの価値を高く，角に隣接するマスの価値を低く設定するのが良いとされている．本研究では，通常のリバーシの評価値マップを元に，Yonin 特有の要素である陣地を利用して変更を加えた評価値マップを使ったAI を作成する．

また，通常のリバーシでは自駒と相手駒が明確に分ることができるため，評価値は，各マスに付与

した値に自駒ならプラスを，相手駒ならマイナスをかけることで求めることができる．しかし，Yonin では自分と同じ色の石を使う相手プレイヤーが存在している．そのため自分の打った石を相手に利用されることや，逆に相手の打った石を自分が利用することができ，盤面上の同色の石を自駒と相手駒に明確に分けることは困難となっている．このことから本研究で使う AI の評価関数では各マスに付与した値に，自色と同じ色の石が置かれていればプラスを，自色と異なる色の石が置かれていればマイナスをかけて評価値を求める．

3.2 AI プログラム

本研究では，先に述べたように陣地を利用した評価値マップを用いた AI を作成する．本研究では，自分の陣地の評価値を高く設定する Self，自分の陣地及び右隣のプレイヤーの陣地の評価値を高く設定する Right，対面のプレイヤーの陣地を高く設定する Opposite，左隣のプレイヤーの陣地の評価値を高く設定する Left，及び陣地による補整をかけない Normal を作成した．付録に本研究で作成したプログラムを示す．また，表 1～5 に今回作成した AI で使用する，左上のプレイヤーにとっての評価値マップを示す．

表 4:Self の評価値マップ

100	-30	0	-1	-51	-50	-80	50
-30	-50	-3	-3	-53	-53	-100	-80
0	-3	0	-1	-51	-50	-53	-50
-1	-3	-1	0	-50	-51	-53	-51
-51	-53	-51	-50	-50	-51	-53	-51
-50	-53	-50	-51	-51	-50	-53	-50
-80	-10	-53	-53	-53	-53	-10	-80

表 3:Right の評価値マップ

100	-30	0	-1	-51	-50	-80	50
-30	-50	-3	-3	-53	-53	-100	-80
0	-3	0	-1	-51	-50	-53	-50
-1	-3	-1	0	-50	-51	-53	-51
-21	-23	-21	-20	-50	-51	-53	51
-20	-23	-20	-21	-51	-50	-53	-50
-50	-70	-23	-23	-53	-53	-10	-80

表 1:Opposite の評価値マップ

100	-30	0	-1	-51	-50	-80	50
-30	-50	-3	-3	-53	-53	-100	-80
0	-3	0	-1	-51	-50	-53	-50
-1	-3	-1	0	-50	-51	-53	-51
-51	-53	-51	-50	-20	-21	-23	-21
50	-53	-50	-51	-21	-20	-23	-20
-80	-10	-53	-53	-23	-23	-70	-50

表 2:Left の評価値マップ

100	-30	0	-1	-21	-20	-50	80
-30	-50	-3	-3	-23	-23	-70	-50
0	-3	0	-1	-21	-20	-23	-20
-1	-3	-1	0	-20	-21	-23	-21
-51	-53	-51	-50	-50	-51	-53	-51
-50	-53	-50	-51	-51	-50	-53	-50
-80	-100	-53	-53	-53	-53	-100	-80

表 5:Normal の評価値マップ

100	-30	0	-1	-1	0	-30	100
-30	-50	-3	-3	-3	-3	-50	-30
0	-3	0	-1	-1	0	-3	0
-1	-3	-1	0	0	-1	-3	-1
-1	-3	-1	0	0	-1	-3	-1
0	-3	0	-1	-1	0	-3	0
-30	-50	-3	-3	-3	-3	-50	-30
100	-30	0	-1	-1	0	-30	100

4 実験結果

先に挙げた 5 種類の局面の評価法を比較するためにそれぞれの AI と，ランダムに打つプログラムと対戦させた．ただし，AI の先読み手数は 4，各試行回数は 100 回である．表 1 に各戦略の対戦の結果を示す.

表 6 より，自分の陣地にのみ補整をかけた Self のみ 1 位回数，最下位回数，平均順位の全てが Normal と比較して良い結果を出している．逆に，右側のプレイヤーの陣地にも補整をかけた Right は全てが Normal よりも悪い結果となっている．対面のプレイヤーの陣地にも補整をかけた Opposite と左側のプレイヤーの陣地にも補整をかけた Left は 1 位回数こそ Normal よりも少ないが，平均順位は Normal よりも高く，最下位回数は Self よりも少ないという結果となった.

表 6:ランダムプログラムとの対戦結果

	1 位回数	最下位回数	平均順位
Self	30	19	2.3 位
Right	20	26	2.6 位
Opposite	21	15	2.4 位
Left	24	17	2.4 位
Normal	28	21	2.5 位

次に，順番による有利不利を検証するために，Self のプレイ順を変えてランダムに打つプログラムを相手に試行した．表 2 にその結果を示す.

表 7 では最初や最後に手を打つよりも 2 番目か 3 番目に手を打つ方がわずかながら良い結果が出ている.

表 7:Self のプレイ順ごとの対戦結果

プレイ順	1 位回数	最下位回数	平均順位
1 番目	30	19	2.3 位
2 番目	33	18	2.2 位
3 番目	36	19	2.2 位
4 番目	30	17	2.4 位

5 考察

第 4 章で示した結果から，通常のリバーシにはない陣地という要素を利用して評価値マップに補整をかけることで，補整をかけてない評価値マップを用いたAIよりも良い結果を出すことができることがわかる．一方で，評価値マップに不適切な補整をかけると結果は悪くなる場合もあると推測できる．また，1 位回数が Normal よりも少ない AI でも平均順位や最下位回数で良い結果を出している AI もあることから，補整のかけ方によっては勝てるプログラムとは別の，通常のリバーシにはなかった負けないプログラムを作成するなどといったバリエーションに富んだAIを作成することも可能だと考えられる．

一方でプレイ人数が増えることで順番による結果の差が大きくなると考えていたが，今回の結果ではプレイ順によってそれほど大きな差は出なかった．それでも多少の差は出ているためより強いAIであればプレイ順によって大きく勝敗に影響が出るとも推測でき，順番によって適切なAIが変わる可能性もある．

6 結論及び今後の課題

本研究では 4 人用に拡張されたリバーシ Yonin の AI を作成した．本研究で作成した AI はどれも強いものではなかったが，評価値マップに陣地による補整をかける点や，2 人プレイと違い勝敗が単なる勝ち負けではなく順位である点を使って様々な特徴のAIを作成することができることがわかった．

今後，より強い AI を作る上で，探索の効率化し，より深く探索を行う必要がある．また，陣地による評価値マップへの補整の改善や，時計回りに手番が回るためベースが通常のリバーシとは異なる評価値マップが対称でなくなる可能性など，評価値マップもまだ改善の余地がある．加えて石の配置による連携や盤面ごとの着手可能数による評価，前半と後半で評価値マップを変更するなどで改善することも可能だと考えられる．

また，今回作成した AI がただ乱数に従って打つだけのプログラム相手にも決して良い結果を出せなかったのはAIの性能以外にも，プレイ人数が増えたことで手番が一周するまでに打たれる手の数が増え，本来想定していた局面とは違う局面になりやすいからなのではと考えている．これに関しては定石の確立や，探索をより深く行うことで改善できる．しかし，想定していた局面とは違う局面になりやすいことでAI同士の相性で勝率が大きく変わる可能性もあると考えている．

Yonin では，自陣の石の数しか勝敗には影響しないので，自分以外の陣地の石の多寡は， 1.4 節で述べた不決定状態になる．不決定状態は解析の難易度を上げる要因となるので，これに対してどう対処するかも今後の課題である．

7 参考文献

[1] 藤井昌典，北隼人，村田朋紀，橋本隼一，飯田弘之：4人版リバーシYonin，情報処理学会研究報告(2006)，http://ci.nii.ac.jp/naid/110004683810

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

[3] 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

[4] 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

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

[6] 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/

[7] 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).

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

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

[10] 最強オセロ「WZebra」，http://homepage3.nifty.com/akky-han/100529.html

[11] MasterReversi Home Page, http://homepage2.nifty.com/t_ishii/mr/index.html

[12] Nipp - アブストラクトゲーム博物館，http://www.nakajim.net/index.php?Nipp

[13] みんなでオセロ，メガハウスのおもちゃ情報サイト，(2013)，http://www.megahouse.co.jp/megatoy/products/item/1139/

[14] 西野順二，西野哲郎，多人数不完全情報ゲームの簡略化評価値による探索を用いた終盤データベースの構築，情報処理学会論文誌 数理モデル化と応用 Vol.3, No.2, pp.11-22, (2010)，http://id.nii.ac.jp/1001/00068410/

[15] 篠原拓嗣，石田亨，N 人ゲームにおける最優良探索，情報処理学会論文誌，Vol.43, No.10, pp.2981-2989 (2002)，http://id.nii.ac.jp/1001/00011454/

[16] Carol.A.Luckhardt and Keki.B Irani, An algorithm solution of N-person games, proc. the Association for the Advancement of Artificial Intelligence , AAAI-86, pp.158-162, (1986). http://aaaipress.org/Papers/AAAI/1986/AAAI86-025.pdf

[17] Nathan.R.Sturtevant and Richard.E.Korf, On prunging techniques for multi-player games, Proc. the Association for the Advancement of Artificial Intelligence , AAAI-00, pp.201-207, (2000). http://www.aaai.org/Papers/AAAI/2000/AAAI00-031.pdf

[18] 橋本剛，前原彰太，川島哲哉，小林康幸，局面評価関数を使う新たな UCT 探索法の提案とオセロによる評価，情報処理学会論文誌，Vol.54, No.7, pp.1930-1936, 情報処理学会，(2013-07-15)， http://id.nii.ac.jp/1001/00094371/

[19] 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(GI)，Vol.2012-GI-27, No.5, pp.1-8, 情報処理学会，(2012)，http://id.nii.ac.jp/1001/00080933/

[20] 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/

[21] 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/
[22] 大筆豊, オセロプログラムの評価関数の改善について, 研究報告ゲーム情報学(GI), Vol.2003-GI-011, pp.15-20, 情報処理学会, (2004), http://id.nii.ac.jp/1001/00058554/

付録

以下に本研究で作成したYoninのAIであるSelfのプログラムを示す．Self以外のAIに関しては評価値の設定以外同一のプログラムとしているので省略する．

```
public class SelfStrategy extends ValueStrategy{

	public SelfStrategy(int player_no){
		setValue(player_no);
	}

	@Override
	public Point decide(Player player) {
		ArrayList<Point> possibles = player.getPossibles();
		Board board = player.board;
		int leng;
		if(possibles.size() > 5){
			int[] vals = new int[possibles.size()];
			for(int i=0; i<possibles.size(); i++){
				Point point = possibles.get(i);
				board.putDisc(point.x, point.y,
player.getColor());
				vals[i] = calculateValue(player,board);
				board.undo();
			}
			possibles = sort(possibles, vals, player);
			leng = 5;
		}else{
			leng = possibles.size();
		}
		int[] vals = new int[leng];
		for(int i=0; i<leng; i++){
			vals[i] = getValue(player, this.depth);
		}
		int max = 0;
		for(int i=1; i<leng; i++){
			if(vals[max] == vals[i]){
				Random r = new Random();
				if(r.nextInt(2)==0){
					max = i;
				}
			}else if(player.getColor() == Board.WHITE){
				if(vals[max] > vals[i]){
					max = i;
				}
			}else{
				if(vals[max] < vals[i]){
					max = i;
```

```
                        }
                    }
                }
                return possibles.get(max);
        }

        @Override
        public void setValue(int player_no){
                for(int i=0; i<Board.Y_SIZE/2; i++){
                        for(int j=0; j<Board.X_SIZE/2; j++){
                                if(player_no ==1){
                                        values[i][j] +=  0;
                                }else if(player_no ==2){
                                        values[i][j] -= 50;
                                }else if(player_no ==3){
                                        values[i][j] -= 50;
                                }else if(player_no ==4){
                                        values[i][j] -= 50;
                                }
                        }
                }
                for(int i=0; i<Board.Y_SIZE/2; i++){
                        for(int j=4; j<Board.X_SIZE; j++){
                                if(player_no ==1){
                                        values[i][j] -= 50;
                                }else if(player_no ==2){
                                        values[i][j] +=  0;
                                }else if(player_no ==3){
                                        values[i][j] -= 50;
                                }else if(player_no ==4){
                                        values[i][j] -= 50;
                                }
                        }
                }
                for(int i=4; i<Board.Y_SIZE; i++){
                        for(int j=4; j<Board.X_SIZE; j++){
                                if(player_no ==1){
                                        values[i][j] -= 50;
                                }else if(player_no ==2){
                                        values[i][j] -= 50;
                                }else if(player_no ==3){
                                        values[i][j] +=  0;
                                }else if(player_no ==4){
                                        values[i][j] -= 50;
                                }
                        }
                }
                for(int i=4; i<Board.Y_SIZE; i++){
```

```
                        for(int j=0; j<Board.X_SIZE/2; j++){
                                if(player_no ==1){
                                        values[i][j] -= 50;
                                }else if(player_no ==2){
                                        values[i][j] -= 50;
                                }else if(player_no ==3){
                                        values[i][j] -= 50;
                                }else if(player_no ==4){
                                        values[i][j] +=  0;
                                }
                        }
                }
        }

        @Override
        public int getValue(Player player, int depth) {
                Board board = player.board;
                if(depth == 0 || board.isgameOver()){
                        int pn;
                        if(player.getPlayerNumber()==1){
                                pn = 4;
                        }else{
                                pn = player.getPlayerNumber()-1;
                        }
                        return calculateValue(board.getPlayer(pn-1), board);
                }else{
                        player.setPossibles();
                        ArrayList<Point> possibles = player.getPossibles();
                        int size = possibles.size();
                        if(size == 0){
                                board.pass();
                                Player pl = board.getPlayer(board.currentPlayer);
                                int val = getValue(pl, depth-1);
                                board.undo();
                                return val;
                        }else{
                                int[] vals = new int[size];
                                for(int i=0; i<size; i++){
                                        Point p = possibles.get(i);
                                        board.putDisc(p.x, p.y,
player.getColor());
                                        Player pl =
board.getPlayer(board.currentPlayer);
                                        int v = getValue(pl, depth-1);
                                        vals[i] = v;
                                        board.undo();
                                }
                                int val = vals[0];
```

```
                        for(int i=1; i<size; i++){
                                if(player.getColor() == Board.WHITE){
                                        if(val > vals[i]){
                                                val = vals[i];
                                        }
                                }else{
                                        if(val < vals[i]){
                                                val = vals[i];
                                        }
                                }
                        }
                        return val;
                }
        }
    }

    @Override
    public int calculateValue(Player player, Board board) {
        int val=0;
        setValue(player.getPlayerNumber());
        for(int i=0; i<Board.Y_SIZE; i++){
            for(int j=0; j<Board.X_SIZE; j++){
                int v = board.getPoint(j, i) * values[i][j];
                val += v;
            }
        }
        return val;
    }
}
```