卒業研究報告書

題目

ニップの並列化

指 導 教 員
石 水 隆 講師

報告者
09-1-037-0171
松 浦 美 里
近畿大学理工学部情報学科

平成24年2月16日提出

概要

本論文は二人零和有限確定完全情報ゲームに分類されるニップについて述べる．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る必要がある．これはニップも同様であり，強い AI を作るためには先読み手数を増やさなければならないが，初手から最終手まで先読みをしようとすれば膨大な時間がかかる．その膨大な時間を減らすために探索を省略する先読み手法もあるが, 探索の省略には限界がある．さらに先読みの計算時間を短縮するにはアルゴリズムの改良によってさらに探索する局面を減らすことは必要であるが, 先読みの計算を複数同時に行うことができればアルゴリズムの改良以上に大幅な時間短縮が見込める．

そこで，本研究ではニップの AI を作成し, その探索を並列化することによってプログラムの高速化を目指す．

目次

1 序論

1.1 背景

ニップ[12]とはリバーシ[2]の一種であり，円形リバーシと呼ばれることもある．通常のリバーシは正方形のゲーム盤上に石を配置するため，角に石を置けるかどうかでほぼ勝敗が決まってしまう．しかしニップでは盤面が円形になっているため，どの石も最後までひっくり返される可能性がある．そのため，リバーシよりも終盤での逆転が起こり易い．

ニップやリバーシ等に代表されるボードゲームは,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常に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数が有限であること．確定とはランダム性が無いこと．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,過去,現在の情報を全てシルことができるゲームである．二人零和有限完全情報ゲームは,その性質上解析を行い易いため,ゲーム理論において様々な研究がなされてきた．また,人工知能の分野においても広く研究がなされている．

1.2 二人零和有限確定完全情報ゲームの完全解析に関する既知の結果

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．チェッカーは双方最善手を指すと引き分けとなる．

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．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,5x5の囲碁は黒の25目勝ちとなる．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．

ニップについては,現在のところ完全解析は行われておらず,サイズを小さくしたミニニップに関しても最適解は出ていない．

1.3 二人零和有限確定完全情報ゲームに対する手法

1.3.1 先読みと局面の評価値

可能な局面数が多いリバーシや将棋といった完全情報ゲームの AI を作成する場合, 数手先の局面を先読みし, 先読み後の局面の評価値によりどの手を打つか決定することが多い．強い AI を作るにはこの先読み手数を多くする必要があるが, 先読みしなければならない局面の数は指数関数的に増えていくため, 処理に膨大な時間がかかってしまう．このため, 先読み処理を並列化することで処理時間の短縮を図ることが多い.

1.3.2 定石データベース・対戦データベース

定石データベースとは,リバーシやニップ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リバーシ・ニッププログラムとなる．しかし,相手があえて定石以外の手を打つなどして,データベースに無い局面が出てきたときにはこの手法は使えない.

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くという手法も考えられる．過去の対戦において,その手が有効であったかどうかを対戦結果から判定し,データベースに蓄える．数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．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リバーシ・ニッププログラムになる．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学習しておく必要がある．しかし,学習が足りなかったり,事前の対戦でなかった手を打たれたりした場合には使えないという欠点がある.

1.3.3 モンテカルロ法

リバーシではあまり使われないが,モンテカルロ法の利用も考えられる．モンテカルロ法とは,各着手可能手に対し,その手から先終局までをランダムに打ち勝敗判定を行うという作業を数千～数万回繰り返し,最も勝率の高い着手可能手を採用するというものである．この手法は局面数が極めて多い囲碁プログラムでは最近主流になっている.

1.4 並列計算

近年,計算機の性能は向上し続けており,ハードディスクの記憶容量なども増加し続けている．計算機の性能向上に伴い,一度に扱うデータの量は増大して来ている．しかし,計算機が扱うデータ量は今後も増大していくと予想される一方,計算機自体の性能の向上は近い将来頭打ちになることが予想される．膨大なデータに対する処理の高速化の方法として,複数のプロセッサを持つ並列計算機(Parallel Computer)を用いた並列処理(Parallel Processing)がある.

並列計算を行うメリットとして指数関数的に増えていく局面の評価計算において複

数のプロセッサに計算を配分できることがある．プロセッサが4つであれば単純計算で各プロセッサが受け持つ計算量は4分の1になり,計算に要する時間が短くなることで高速化となる．現在並列計算は計算量が膨大となる新薬の開発[16]や気象現象の解明[17]などで広く使われている．

並列計算を行ううえで,通常の計算では気にする必要が無かったが,並列計算では注意しなければならないこともある．並列化しても,各プロセッサにほぼ均等に計算量が割り当てられなければ並列化によって短縮される時間はわずかであるし,計算結果をその後使用するのであれば次の段階に進む前に全てのプロセッサが計算を完了するのを待たなければならない．こういった問題を無視するとデータのハザードやデッドロックが発生する可能性がある[18].

1.5 既存のニップ AI

リバーシであれば「WZebra[13]」や「MasterReversi[14]」,チェスであれば「Deep Junior[15]」などが強い AI として有名であるが,ニップには強い AI として有名になっているものが無い．既知のニップ AI としては「OpenNIP[3]」があるが、現在のところ、それほど強いという評価は受けていない。

1.6 本研究の目的

1.3節で述べた通り,多くのゲームでは強いAIを作るためには先読み手数を大きくする必要がある．これはニップも同様であり，強いAIを作るためには先読み手数を増やさなければならないが，それには膨大な時間がかかる．そこで，本研究ではニップの探索を並列化することによってプログラムの高速化を目指す.

1.7 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,本研究が対象とするニップについて説明する. 続く第3章で,ニップ AI が打つ手を決定する手法について述べる. そして4章において作成した AI を,候補手をランダムに打つ AI や作成した AI 同士による対戦などの結果を報告する. 第5章において第4章の結果について考察し第6章において結論および今後の課題を述べる.

2 準備

2.1 ニップについて

まず，ニップというゲームについて簡単に解説する．先程も述べたように，ニップは円形リバーシと呼ばれることもあり盤面の外周部が円形になっている以外は基本的な部分はリバーシと同じである．図1にニップのゲーム盤と,石の初期配置を示す．ニップとリ

バーシで最も大きな違いは外周部である．リバーシであれば外周部は角と辺であり,角とそれに隣接する辺に石を置くことができれば,その石はゲーム終了時までひっくり返されることが無い確定石とすることができるが，ニップでは外周部の石が無条件に確定石とはならない．例えば図2のように外周のほとんどを黒が押さえた時，白が図3と打つことで図4となる．このように外周部をほぼ押さえたからといって必ずしも有利とはならず，最後まで勝負がわかりにくいゲームとなっている．

また,ニップは,石をひっくり返す順番にも気を付けねばならない．通常のリバーシであれば,置いた石から縦横斜め方向のどの石からひっくり返しても結果は同じである．しかしニップでは,円周上のマスに石を置いた場合,縦横斜め方向を先にひっくり返した場合と,円周方向を先にひっくり返した場合とで結果が変わってしまうことがある．つまり,円周方向を先にひっくり返すことにより,縦横斜め方向の突き当りの石が自分の石になり,その結果間の石がひっくり返されてしまう．ニップもリバーシと同様に,どの石をひっくり返せるかは石が置かれた時点で判定を行わなければならないので縦横方向からひっくり返さなければならない．

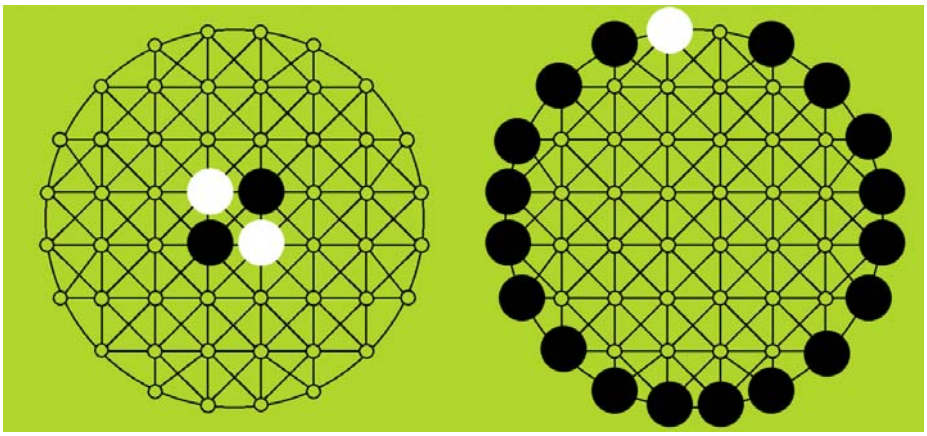

図1：ニップの盤面および駒の初期配置　　図2：外周への石置き例(1)

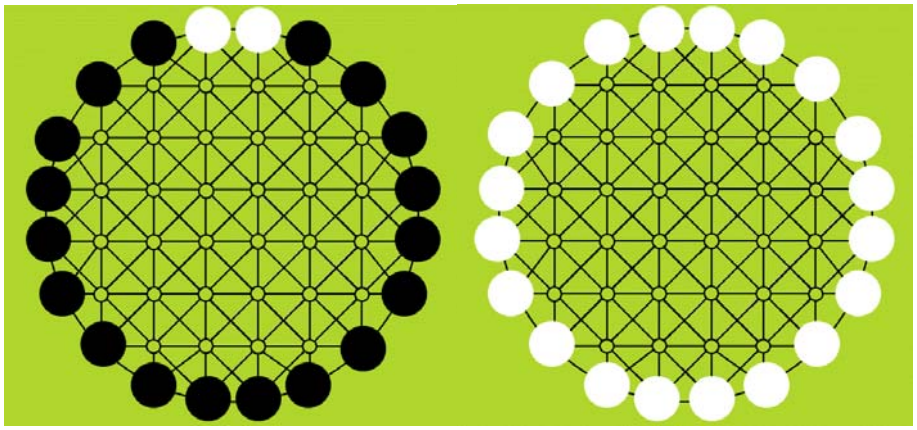

図3：外周への石置き例(2)　　図4：外周への石置き例(3)

2.2 ニップの局面数

本節ではニップで可能な局面数について考える．ニップ可能な局面数はおよそ10^{24}通りである．完全解析のされているサイズ6x6のリバーシであれば可能な局面数はおよそ10^{17}であり,完全解析を行うには少々多いように思われる．

3　研究内容

本研究では,まず逐次のニップ AI を作成する．

3.1 ニップ AI で用いた手法

1.3節で述べたように,次に打つべき手をどのように選択するかは様々な手法がある．本節ではニップ AI で用いた手法について説明する．作成した AI では,現在の盤面からの候補手の評価値からその候補手を打った場合の相手の候補手の評価値を引き算し,その相手の候補手からの自分の候補手の評価値を足し算して最も値の大きい候補手を採用するという手法を採った．

3.1.1 着手可能手の決定

ある局面において着手可能手を探索するには,まず現在の手番となっているプレイヤーから見て相手の色の石を探索する．その周囲のマスを探索し,同じ色が続く限りはその方向に探索を続け,違う色が出てきたら探索を中止し,探索の出発点の探索してきた方向と逆方向が空いているマスであれば true を返し,途中で盤外に出たら false を返す．これを全ての石に対して実行する．通常のリバーシであれば,探索する方向は縦横斜めの八方向であるが,ニップの場合,円周上のマスは縦横斜めに加えて円周方向にも探索を行う．2.1節で述べた通りこの探索は縦横方向から先に行う．

3.1.2 評価関数について

本研究で作成したニップ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, 先読みで得られた局面の評価値を用いて手を決定する．

局面の評価の代表的な手法としては，評価値マップの使用がある．評価値マップとは，各マスに正負の価値を付加し，マスに自石が置かれた場合その価値を足し相手石が置かれた場合その価値を引く，というものである．リバーシの場合は角のマスを高価値とし，角に隣接するマスを低価値とすると良いことが一般に知られている．一方，ニップは各マスにどのような価値を付加するかは確立されていない．そこで本研究では，各マスに図5に示す価値を付加し，その有用性を検証する．

リバーシでは一般に，相手の選択肢を減らす手が良いとされている．そこで本研究の評価関数は図5のような評価値マップによる局面評価に加えて表1に示すような相手の選

択肢の数を用いた.

また,一般に打てる候補手の数が多いほど選択の余地があり,有利と考えられる．従って,ある局面の評価値は,その局面で指せる候補手の数も考慮する.

本研究で作成したニップ AI で使用する局面の評価値は以下の式で与えられる.

評価値＝評価値マップによる評価値＋相手の選択肢の数による評価値

		30	30	30	30		
	30	-10	-10	-10	-10	30	
30	-10	-2	3	3	-2	-10	30
30	-10	3	0	0	3	-10	30
30	-10	3	0	0	3	-10	30
30	-10	-2	3	3	-2	-10	30
	30	-10	-10	-10	-10	30	
		30	30	30	30		

図5．評価値マップ

表1．相手の選択肢の数による評価値

相手の選択肢の数	評価値
0	1000
1	100
2	50
3	20
4以上	0

3.2 ニップ AI プログラム

本節では,ニップ AI プログラムについて述べる．付録1に,ニップ AI プログラムのソースを示す.

また、作成した AI クラス以外のニッププログラムは OpenNip プロジェクト[3]にて公開されているソースコードを使用した。

3.2.1 クラス AIStrategy

AIStrategy クラスは、その名の通り AI の戦略を決定するクラスである。この AI にて使用される評価マップの評価値は map[x][y] にて決定される。executeMiddlePhaseStrategy メソッドは評価マップと相手の選択肢の数を用いて評価値を計算,その結果最高の評価を得た手を次の手として配列 bestWays に格納する.evaluateState メソッドは評価マップによる評価を、evaluateNumOfAltanatives メソッドは相手の候補手の数による評価値をそれぞれ返すメソッドである。

4　実験結果

本研究で作成したニップ AI の性能を検証するため、先読み手数を4とし、ランダムで打つ AI および既存のニップ AI[3]との対戦を先手後手それぞれ100回行った．表2に対戦結果を示す．

表2より，ランダムには先手で7割の確率で勝ち、後手でも6割の確率で勝利している．また、先手のほうが勝率が上がっていることが分かる．既存のニップ AI には、先手後手両方とも六割強の確率で勝利している．

表2．対戦結果（試行回数100回）

対戦	先手勝ち	後手勝ち	引き分け
AI 対ランダム	75	25	0
ランダム対 AI	38	62	0
AI 対[3]	67	33	0
[3]対 AI	32	68	0

次に，評価値の重みを表3のように評価値マップによる評価よりも相手の選択肢の数を重視するよう変えて同様に先読み手数を4として実験を行った．表4に結果を示す．ランダムとの対戦において先手では8割ほどの確率で勝利しており， 表2よりも勝率が少し上がったことが分かる．後手の場合は表2よりも少し下がってしまっている．既存のニップ AI[3]との対戦結果は先手にはほとんど変化は見られない，が後手の場合は表2よりも一割ほど下がってしまっている．

表3．　相手の選択肢の数による評価値

相手の選択肢の数	評価値
0	1000
1	500
2	100
3	50
4	10
5以上	0

表4．対戦結果（試行回数100回）

対戦	先手勝ち	後手勝ち	引き分け
AI 対ランダム	81	19	0
ランダム対 AI	31	69	0
AI 対[3]	64	36	0
[3]対 AI	42	58	0

5　考察

実験結果から今回作成した AI はランダムとの対戦結果において，表3の結果より表4の結果のほうが先手後手ともに勝率が上がっていることから評価値マップによる評価よりも相手の選択肢の数を重視するように変えると勝率が上がると考えられる．また既存のニップ AI に対しても表3では先手後手ともに六割強の勝率で、表4でもほぼ六割の勝率で勝ち越していることから，作成した AI は，ランダムや既存のニップ AI よりも優れていると考えられる．

今回作成した AI ではそれぞれの候補手による評価値を毎回加算しているが,この方法では途中まで優勢でも最後に逆転されるというような予想になっても,途中が優勢であったため最終的には劣勢になっているにも関わらず評価値が高くなってしまい,悪手であるはずの手が最善手として選ばれてしまう．

勝率を高める他の方法として,序盤は統計的に勝率の高い手を打つことや,終盤は必勝読みや完全読みに切り替えることが挙げられるニップがリバーシと違い,外周の石も全てひっくり返せることを考えると,相手の確定石を外周に作らせない,あるいは自分の確定石を外周に作ることも評価に入れることも考えられる．

6　結論および今後の課題

本研究では,ニップ AI を作成した．本研究で作成したニップ 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, 先読みで得られた局面の評価値を用いて手を決定する．評価値には評価値マップによる評価と相手の選択肢の数を用いた．作成した AI はさほど強いものではなかったが, ニップの評価マップにおいて,外周を重視すればある程度は強くなったものの,決定的に強くなったわけではないということを示すことはでき,今後のニップの研究に生かすことができると思う．

本研究では,評価関数についての実験報告という段階にとどまってしまった．また,本研究ではニップの並列化を目指してはいたため,評価関数の強化や定石の適用については特に考えていなかった.

今後,並列化とともに強い AI を作るための評価関数の研究を行っていきたい．今後の課題としては、以下のことが挙げられる.

1 ニップにおける評価関数においては相手の選択肢の数は重要ではない可能性があり,さらなる検証が必要である.

2 評価マップによる評価が適切であったどうか疑問であり,検証が必要である.

3 途中まで優勢であっても最後に外周を全てひっくり返され,逆転されることがあり,終盤になれば戦術を評価マップによるものから必勝読みに切り替える必要がある.

4 本研究において最終目標としていた並列化についてはできておらず,評価関数の改善と共に必要なことである.

参考文献

[1] 結城浩：Java 言語で学ぶデザインパターン入門【マルチスレッド編】, ソフトバンククリエイティブ, (2006)

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

[3] OpenNip(オープンニップ) プロジェクト, http://sourceforge.jp/projects/opennip/

[4] 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

[5] 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

[6] 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

[7] 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/

[8] 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).

[9] 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/

[10] 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/

[11] 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).

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

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

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

[15] 14th 世界コンピュータチェス選手権, http://www.grappa.univ-lille3.fr/icga/tournament.php?id=16&lang=3

[16] 「63年かかる分子動力学計算を2週間で」富士通の超並列サーバ, (2003), http://www.atmarkit.co.jp/news/200311/06/fujitsu.html

[17] 地球シミュレーター http://www.jamstec.go.jp/es/jp/index.html

[18] 第1回並列ゼミ 並列処理概論 - 知的システムデザイン研究室, (2000), http://mikilab.doshisha.ac.jp/dia/seminar/2000/parallel_1.pdf

付録

以下に本研究で作成したニップ AI のソースを示す。

```
public class AIStrategy implements NipStrategy {
    private Random rand = new Random();
    private int[][] map = {{99, 99, 30, 30, 30, 30, 99, 99},
                           {99, 30, -10, -10, -10, -10, 30, 99},
                           {30, -10, -2,  3,  3, -2, -10, 30},
                           {30, -10,  3,  0,  0,  3, -10, 30},
                           {30, -10,  3,  0,  0,  3, -10, 30},
                           {30, -10, -2,  3,  3, -2, -10, 30},
                           {99, 30, -10, -10, -10, -10, 30, 99},
                           {99, 99, 30, 30, 30, 30, 99, 99}};

    private int depth;
    private long waitTime;
    private class State{
        private NipTable table;
        private NipStone stone;
        private State(NipTable table, NipStone stone) {
                this.table = table;
                this.stone = stone;
                }
    }
    public AIStrategy(int depth, long waitTime) {
        this.depth = depth;
        this.waitTime = waitTime;
    }
@Override
    public int[] decide(NipTable table, NipStone stone) {
        int[] decision = executeMiddlePhaseStrategy(table, stone, depth);
        return decision;
        }
    private int[] executeMiddlePhaseStrategy(NipTable table, NipStone stone,int depth) {
        ArrayList<int[]> list = (ArrayList<int[]>)
```

```
NipTableUtil.getCanPutStoneCellList(table, stone);
      ArrayList<int[]> bestWays = new ArrayList<int[]>();
      int bestScore = -10000;
      ArrayList<Integer> scoreList = new ArrayList<Integer>();
      scoreList.clear();
      for(int[] index : list) {
                NipTable state = NipTableUtil.ifYouPutStoneAt(table, index[0], index[1],
stone);
               int score = evaluateState(state, stone) + evaluateNumOfAltanatives(state,
stone);
                scoreList.add(score);
                for(int i=0;i<5;i++){
                          List<int[]> list2 = NipTableUtil.getCanPutStoneCellList(table,
stone);
                for(int[] index2 : list2){
                                    NipTable state2 = NipTableUtil.ifYouPutStoneAt(table,
index2[0], index2[1], stone);
                                    int score2 = evaluateState(state2, stone) +
evaluateNumOfAltanatives(state2, stone);
                                    int scoreTotal=-1000;
                                    List<int[]> list3 =
NipTableUtil.getCanPutStoneCellList(table, stone);
                                    for(int[] index3 : list3){
                                              NipTable state3 =
NipTableUtil.ifYouPutStoneAt(table, index3[0], index3[1], stone);
                                              int score3 = evaluateState(state3, stone) +
evaluateNumOfAltanatives(state3, stone);
                                              List<int[]> list4 =
NipTableUtil.getCanPutStoneCellList(table, stone);
                                              for(int[] index4 : list4){
                                                        NipTable state4 =
NipTableUtil.ifYouPutStoneAt(table, index4[0], index4[1], stone);
                                                        int score4 = evaluateState(state4,
stone) + evaluateNumOfAltanatives(state4, stone);
                                              scoreTotal = score - score2 + score3 -
score4;
```

```
                                        scoreList.add(scoreTotal);
                                        if(scoreTotal > bestScore) {
                                                bestWays.clear();
                                                bestWays.add(index);
                                                bestScore = scoreTotal;
                                        } else if(scoreTotal == bestScore) {
                                                bestWays.add(index);
                                        }
                                }
                        }
                }
        }
    }
        return bestWays.get(rand.nextInt(bestWays.size()));
    }
    private int evaluateState(NipTable table, NipStone stone) {
        return NipTableUtil.evaluate(table, stone.getColor(), map);
    }
    private int evaluateNumOfAltanatives(NipTable table, NipStone stone) {
        int numOfAltanatives = NipTableUtil.getCanPutStoneCellCount(table,
    NipTableUtil.differentColorStone(stone));
        int value;
        switch(numOfAltanatives) {
//パターン１
/*      case 0:
                value = 1000;break;
        case 1:
                value = 100;break;
        case 2:
                value = 50;break;
        case 3:
                value = 20;break;
        default:
                value = 0;
        }
        return value;
```

```
*/
//パターン２
        case 0:
                value = 1000;break;
        case 1:
                value = 500;break;
        case 2:
                value = 100;break;
        case 3:
                value = 50;break;
        case 4:
                value = 10;break;
        default:
                value = 0;
        }
        return value;
    }
}
```