卒業研究報告書

題目

モンテカルロ法によるニッププログラム

指導教員

石水　隆　講師

報告者

10-1-037-0020

中村　佳亮

近畿大学理工学部情報学科

平成 26 年 1 月 31 日提出

概要

本論文は二人零和有限確定完全情報型ボードゲームに一種であるニップについて述べる。二人零和有限確定完全情報型ボードゲームは、理論上は、起こりえる全ての局面を解析することにより必勝手順を求めることができる。しかし、多くのゲームでは、可能な局面数は膨大であり、現在の計算機性能では完全解析は不可能である。しかしながら、完全解析は不可能だろしても、多くのゲームで様々なよりよい手を選択する方法が提案されている。

本研究では、その様々な着手の選択方法の中からモンテカルロ法を採用し、ニッププログラムの作成を行う。モンテカルロ法とは[6]、乱数を用いたシミュレーションを何度も繰り返し試行する事によって近似解を求める計算手法である。解析的に解く事が難しい問題でも、十分に試行を重ねる事により、近似的に解を求める事ができる。適用範囲が大きく、これがボードゲームの解析にも有効である。しかし、高い精度を得ようとすれば、試行回数を増やさなければならないので、計算処理が膨大になってしまうという弱点もある。

目次

1　序論 4

1.1　背景 4
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 4
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の解析に対する手法　4
1.4　リバーシ・ニップに関する既存の AI 5
1.5　本研究の目的 5
1.6　本報告書の構成 5

2　準備 5

2.1　ニップについて 5
2.2　ニップのルール 6
2.3　リバーシとの相違点 7
2.4　ニップの局面数 7

3　研究内容 7

3.1　着手可能手の~~決定~~探索 7
3.2　着手可能手の選択 8
3.3　ニップ AI プログラム 9

4　結果・考察 10

5　結論・今後の課題 11

参考文献 14

1 序論

1.1 背景

ニップ（Nip）とは[3]、リバーシとよく似た盤上ゲームである。リバーシとの相異点は、正方形の盤ではなく円形の盤を使用することである。ニップは、円形の盤を使用するためリバーシでいう角という確定石がなくゲーム終盤でも十分に逆転する事が可能となっているのが特徴である。

ニップやリバーシ等のボードゲームは、二人零和有限確定完全情報ゲームに分類される。二人零和有限確定完全情報ゲームとは、以下に挙げる条件を満たすゲームである。

- 零和・・・プレイヤーの利得の合計が常に 0 になることである。
- 有限・・・プレイヤーが指す手の組み合わせ数が有限であることである。
- 確定・・・ランダム要素が無いことである。
- 完全情報ゲーム・・・各プレイヤーが自分の手番に双方の手を含め過去、現在の情報を全て知る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。

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とは、両者が最善手を指し続ける事で先手・後手必勝か引き分けのどれかがゲーム開始時点から決まっているゲームの事を意味する。すべての着手可能手を解析すれば理論上最善の手が打てるが、多くのボードゲームは、解析しなければならない着手可能な局面での数が多すぎて、完全解析に至らない。一例として、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} 通りとなっている。これは現在の計算機の性能を越えている値となっている。一方、可能な局面数が少ないゲームでは完全解析がされているものもある。連珠は双方が最善手を指し続けると 47 手で先手必勝となる[8]。チェッカーは両者が最善手を指し続ければ引き分けになる[9]。バンダイが開発した「シンペイ」は最長 49 手で、後手必勝となる[10]。このように完全解析がされてきている。また、局面数が大きいゲームについては、ゲーム盤を小さいサイズに限定した場合の解析も行われている。本研究でも本来 16 路盤を用いるハルマを 8 路盤に限定して研究を行っている。完全解析されて例として、6 路盤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[11]。将棋では、盤のサイズが 3×4 で駒の種類が 4 つの「どうぶつしょうぎ」[12]が双方最善手を指すと 78 手で後手勝ちとなることが判明している[13]。囲碁ではサイズ 4x4 のミニ囲碁では待碁に、サイズ 5x5 のミニ囲碁では先手の 25 目勝ちになる[14][15]。

ニップに関する完全解析は、現在のところ行われておらず、盤面のサイズの小さくしたニップに関しても完全解析はできていない。

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の解析に対する手法

可能な局面数が多いリバーシや将棋といった二人零和有限確定完全情報ゲームの AI を作成する場合、様々

な手法をもって解析する事が可能となっている。一般的な解析は数手先の局面を先読みして、先読み後の局面の評価値によって次の一手を決定する事が多い。先読み手数を多くすればするほど強い AI は作れるのであるが、その反面で先読み数を伸ばす事で、先読みする局面が指数関数的に増え、処理時間に膨大な時間を要する。そこで先読み処理を並列化する事で処理時間の問題点を解決できる。またゲームを解析する中で有名な他の手法として、定石データベースや評価関数を用いる手法がある。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て、各局面で有効な定石があればその通りの打ち方をする手法である。定石データベースを使用する事で、AI にゲームに関する知識が増えると同じで強くなる。しかし、定石以外の手を打たれると違う手法を用いて次の一手を打たなければならない。また過去に繰り返し対戦を行っている場合は、過去の対戦記録をデータベース化しておいて、有効な手があれば記憶しておき、そこから打つ手を決める手法もある。これは対戦回数を増やす事で精度が増していく。

上記に述べたもうひとつ手法である評価関数について説明する。評価関数とは、着手可能手が複数ある場合、その手を打った場合に得られる局面を先読みして、それによって得られた局面の評価値を用いて次の一手を決める手法である。局面に対する代表的な手法に評価値マップがある。評価値マップとは、各マスに価値を付加して、あるマスに自石を置いた場合にはその価値を足し、逆に相手に置かれた場合はその価値を引くというものである。

1.4 リバーシ・ニップに関する既存の AI

リバーシで作成されている有名な既存の AI は「WZebra[4]」や「MasterReversi[5]」などが挙げられる。ニップの既存の AI としては「OpenNIP[2]」が挙げられるが、現在のところは強いという評価を受けている AI は存在していない。また、松浦と井藤により評価値マップを用いたニップ AI[16][17]が作られてはいるが、これらはあまり深く先読みをしておらず、既存の OpenNip と比べてもそれほど強くはない。

1.5 本研究の目的

上記に述べた通り、ニップの完全解析はまだ行われていない。よっていきなり完全解析するのは容易ではないため、完全解析の前準備として、強い AI を作成し、その動作を検証することで完全解析に近づけていく。また、強い AI を作成するために先読み手数を伸ばすと莫大な時間がかかってしまうので、本研究では着手探索の手法としてモンテカルロ法を用い、より強い AI を作成する事を目的とする。

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、二人零和有限確定完全情報ゲームの一種であるニップについて説明する。第 3 章は、ニップ AI で使用した解析の手法について述べる。第 4 章では、作成した AI の実証結果についての報告をする。第 5 章では、第 4 章の結果や考察と今後の課題について述べる。

2 準備

2.1 ニップについて

ニップについて、ルールを簡単に説明する。1.1 節に述べた通りニップとは円形の盤面のリバーシである。図 1 にニップのゲーム開始時の初期盤面を示す。基本的なルールはリバーシと同じである。

2.2 ニップのルール

ニップの基本的なルールは以下の通りである。

- 52 マスの円形の盤面を使用し、ゲーム開始時は黒と白の石を中央に2個ずつ対角に石をそれぞれ配置する。図1に盤面を示す。
- 黒が先手で対局が開始される。
- 石を置ける場所は、必ず相手の石を1個以上挟める場所にしか置くことができない。石を置き、挟んだ石は裏返して自分の色の石へと変える。
- 一方が打てる場所がない場合はパスとなり相手のターンへと変わり、打てる箇所ができるまで相手のターンが続く。
- 盤面上のすべてのマスが埋まる、もしくは双方がパスをせざるをえない状況の時は対局が終了となり、盤面の石の色が多い方が勝者となる。双方がパスとなる状態を図 2 に示す。図 2 の局面において黒の番であった場合、黒が星の部分に打つとすべての石が黒色となり、双方が置ける場所がなくなる。また、盤面上のすべてのマスが埋まった状態を図3に示す。

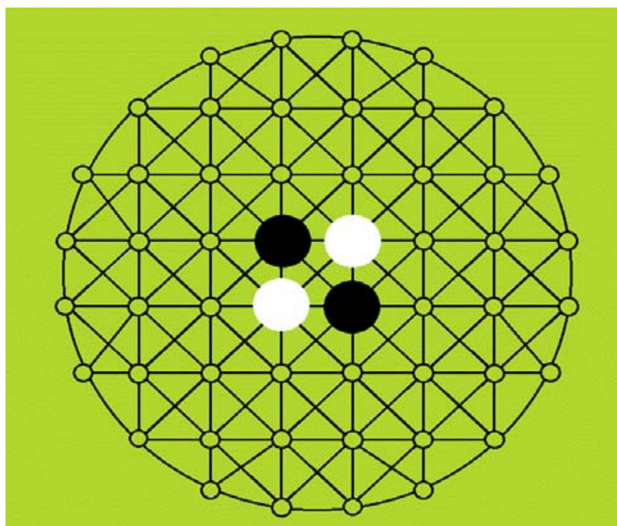

図 1.初期盤面

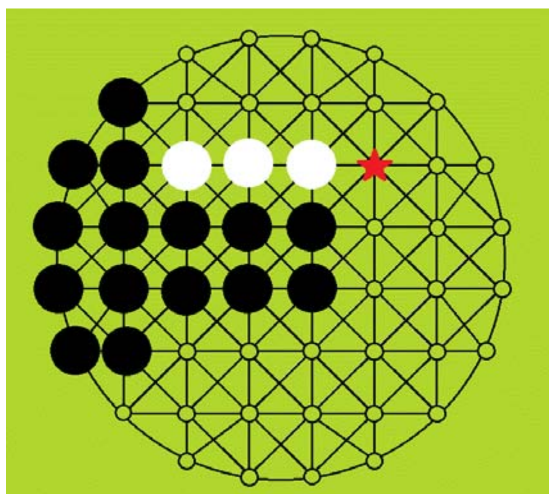

図 2.双方がパスとなる場合

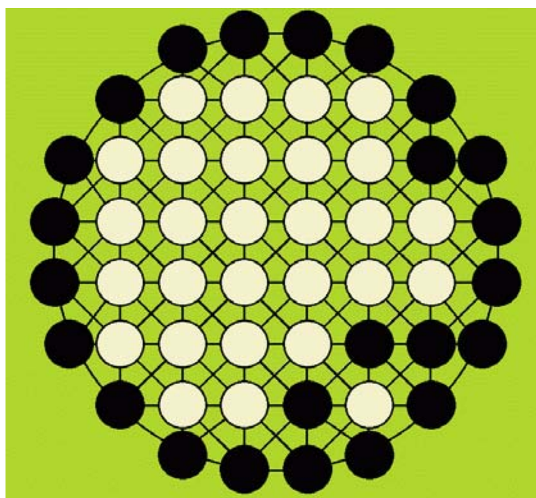

図 3.盤面がすべて埋まった状態

2.3 リバーシとの相違点

ニップとリバーシでの大きな相違点は、盤面の外周のマスである。リバーシには角マスという存在があり、このマスが確定石となる。確定石とは、そのマスに石を置けた時点でその石の色がゲーム終了時までひっくり返される事が無い石の事をいう。一番端の辺に対しても、一辺がすべて同色の石で埋める事ができれば、ひっくり返る事がなくなるという利点がある。それに対してニップは、外周が円となっているために角が存在しない事で無条件による確定石が存在しない。

ニップは円形といった特殊な盤面によってたとえ相手が外周部に石を打ったとしても、最終的に２マス分石を置くだけで外周部のマスのすべてをひっくり返す事が可能となっている。よってゲーム終盤までどのマスもひっくり返せる事が可能となっているボードゲームである。

2.4 ニップの局面数

本節ではニップの盤面にて可能な局面数について示す。ニップの可能な局面数は約 10^{23} 通りである。一方、すでに完全解析がされている 6x6 のミニリバーシでは可能な局面数は約 10^{17} 通り、現在まだ完全解析されていない 7x7 のミニリバーシの可能な局面数が約 10^{23} 通りである。よってニップの完全解析を行うには、局面数が多過ぎるため現在の計算機性能では難しい。

3　研究内容

本研究では、ニップゲームに対する AI を作成し、それについて述べていく。

3.1 着手可能手の探索

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石を探索する。その周辺のマスを探索し、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 の値を返し、途中に盤外にはみでるとなれば false を返す。これをすべての石に実行する。ニップはリバーシと違って盤面が円形のため縦横斜めに円周方向の探索もおこなっている。

3.2 着手可能手の選択

本節では、前節で探索した着手可能手のうちどの手を採用するのか、その戦略について述べる。

リバーシでは、評価値マップによる評価値計算や定石データベース、対戦データベース、終盤での必勝読みといった手法により、強いとされる手を選択することができる。リバーシでは、評価値マップで角マスに置かれた石を高評価、角マスに隣接するマスに置かれた石を低評価にすることにより、ある局面に対してそれなりに適切であると考えられる評価値を得ることができる。一方ニップでは、角が存在しないため、適切な評価値を持つ評価値マップを作ることが難しい。また、リバーシの定石は古くから検証されており、対戦も数多く行われているためデータベースが充実している。対してニップはマイナーなゲームであるため、データの蓄積が充分ではなく、有力な定石も存在しない。これらの理由から序盤から中盤においては、ニップでリバーシと同様の戦略を採ることは困難である。そこで本研究では、序盤から中盤における戦略としてモンテカルロ法[6]を採用する。

モンテカルロ法とは盤上ゲーム等の AI でよく使われている手法の一つである。乱数を用いてシミュレーションを行う手法であり、ゲームの他にも物理的な現象の推測や経済的な分野等、様々な分野で広く用いられている。

モンテカルロ法による手の決定は以下の手順で行われる。ある局面で、その盤面における着手可能手の一つ対して、その手を一手打った後そのままランダムに終局までその勝敗と求める、という操作を一定回数繰り返し、勝率を求める。同様に、この操作を他の全ての着手可能手に対しても行い、それぞれの勝率を求める。そして、最も勝率の高い手を選択する。

以下に、ニップで用いるモンテカルロ法の説明を簡単にする。まず図 4 に示されている、赤色の星マークの部分は自身が黒石のプレイヤーである場合の着手可能手である。この場合は 5 か所が着手可能となっており、5 か所のうちから一点を選ぶ。今回は着手可能手のうちの一番左側を選んだとする。選んで打った後の盤面を図 5 に示す。そのマスに打ったと仮定したその後の展開を、乱数を用いてランダムに双方が終局まで打ち続けるシミュレーションを一定回数試行する。その時の勝敗を評価値として付ける。本研究では一局のシミュレーション毎に勝ちに+1、負けに-1、引き分けに 0 点とつけてその合計を評価値とする。この動作を残りの着手可能手 4 か所でも試行する。すべての着手可能手の試行が終われば、着手可能手の中から最高の評価値がついているマスを次の一手とする。評価値の計算は以下の計算法となっている。

評価値 = (+シミュレーションによる勝ち数) + (−シミュレーションによる負け数)
+ (引き分けの数)

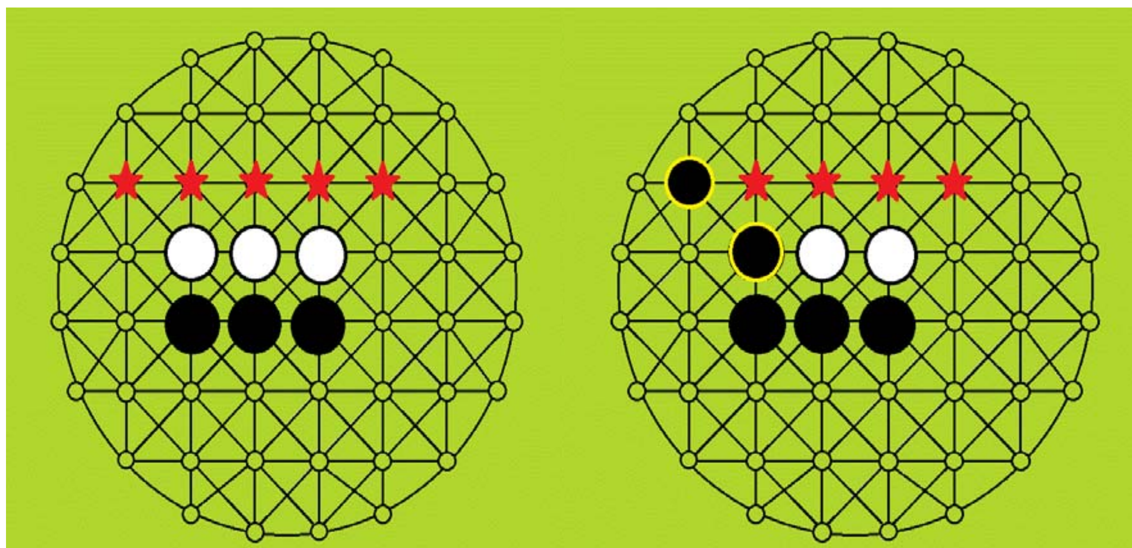

図 4.着手可能手例　　　　　　　　図 5.打ったと仮定した後の盤面

本研究で作成した AI は残り 16 手となるまではランダムで手を打ち、その後以降はモンテカルロ法を用いて着手手を決定する。最初から残り 16 手までをランダムに打つ仕様にした理由は、ニップ確定石となるマスがなく終盤での逆転が多いため、序盤からモンテカルロ法を使用してもあまり有効ではないためである。

3.3 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。本研究では AI クラス以外の部分は OpenNip プロジェクト[2]のソースを少し変更し、作成した。また評価値マップによる AI プログラムは、昨年のニップの研究[16][17]を引き継いで用いている。よって、AI 部以外のプログラムは OpenNip プロジェクト[2]の物を使用した。付録 1 に本研究で作成したクラスである AIStrategy のソースプログラムを示す。また、付録 2 に、AIStrategy クラスの対戦相手となった評価値マップを用いた AIStrategy2 のソースプログラムを示す。

3.3.1　AIStrategy クラス

AIStrategy クラスは、AI の戦略を決定するクラスである。この AI の valueList に着手可能手の評価値をそれぞれ格納している。bestMove メソッドで、valueList を用いて最も評価の高い手を決定する。
表 1 に AIStrategy クラスのメソッドを示す。

表 1 AIStrategy クラスのメソッド

メソッド	処理内容
AIStrategy（long waitTime）	コンストラクタ
int[] decide(NipTable table, NipStone stone)	着手可能手の中から次に打つ一手を選ぶ
int[] bestMove(int[] valueList)	モンテカルロ法によって次に打つ一手を選ぶ
Int[] playOut(NipTabele table,NipStone stone)	乱数によるシミュレーションを行う

4　結果・考察

本研究で作成したニップ AI の性能を検証するため、最初にモンテカルロ法を用いたニップ AI とランダムで打つ AI を先手後手それぞれ 100 回ずつ試行した。その対戦結果を表 1 に示す。上記にも示した通り黒石が先手となっている。

表 1 より、ランダム相手に対しては先手で 8 割、後手で 7 割ほどの勝率となっている。序盤までのランダムでの打ち合いは変わらないにしても、後半以降のモンテカルロ法を用いた打ち方と単純にランダムで打つ事で差がついている事がわかる。この結果により、少なからず作成した AI はランダムで打つ AI よりは強い事が検証された。

表 1.　試行回数 100 回の対戦結果（対　ランダム）

黒対白	黒勝ち	白勝ち	引き分け
AI 対ランダム	81	19	0
ランダム対 AI	28	71	1

次に、着手可能手数および評価値マップにより打つ手を決定する AI（以下、Nipevt とする）[16][17]と、モンテカルロ法を用いた AI との対戦を、先手後手それぞれ 100 回ずつ試行してみた。Nipevt は盤面に与えた評価値を、外周部が高評価になるように評価値を与えた。先読み手数は 4 としている。また Nipevt は、相手が次に打てる選択肢が少なくできるマスにも評価値を与えた。これはリバーシでは相手の打てる選択数を減らす打ち方が良いとされており、ニップでも同様と考えられるためである。従って、Nipevt の評価値の計算方法は、評価値マップによる評価値と相手の選択肢の数による評価値を加算した合計となっている。Nipevt は評価値から最も高い評価値のマスを次の一手と決定する。 盤面の評価値を図 6 に、相手が次に打てる選択肢を減らせるマスの評価値の値を表 2 に示す。また、モンテカルロ法を用いた AI と評価値マップの対戦結果を表 3 に示す。　表 3 の対戦結果より、モンテカルロ法を用いた手法は先手勝ちが約 6 割強となり、後手勝ちが 5 割強となった。モンテカルロ法を用いた手法の方が、評価値を用いた手法より若干であるが上回っている事がわかった。評価値を用いた手法より勝率が若干の差しか出ていない事から、一概にどちらが強いとは言えない結果となってしまった。今回作成したモンテカルロ法は残り 16 手目までランダムで打つ仕様としていたため、その分序盤に明らかに無意味な打ち方をしているケースがあった。よって序盤をランダムで打つのではなく、リバーシで通用する定石等を組み込んでいくと更に強い AI ができるのではないかと考えられる。またモンテカルロ法によるシミュレーションの回数を 100 回で設定していたのだが、もっと試行回数を増やせば更に強い AI も作成できるが、試行回数を増やすと処理時間が増える新たな問題が出てくるので、並列化を実現できればより AI を強くする事も可能であると考えられる。

		100	100	100	100		
	100	-20	-20	-20	-20	100	
100	-20	-20	50	50	-20	-20	100
100	-20	50	0	0	50	-20	100
100	-20	50	0	0	50	-20	100
100	-20	-20	50	50	-20	-20	100
	100	-20	-20	-20	-20	100	
		100	100	100	100		

図 6.盤面の評価値

表 2.　相手の選択肢に対する評価値

相手の選択肢の数	評価値
0	10
1	5
2	3
3	1
4 以上	0

表 3.　試行回数 100 回の対戦結果（対　評価値マップ）

黒対白	黒勝ち	白勝ち	引き分け
AI 対評価値マップ	63	37	0
評価値マップ対 AI	43	55	2

5　結論・今後の課題

本研究では、ニップ AI プログラムを作成した。本研究で作成したプログラムは、ある局面での着手可能手のすべてに着目し、すべての着手可能手にモンテカルロ法を用いてシミュレーションを行い、その結果から出た評価値の一番高いマスに打つ手を決定する。評価のつけ方は勝ち、負け、引き分けをそれぞれ+1、-1、0 としてその合計を評価値として付けた。

作成した AI は評価値マップを用いたものより若干ばかり強いという結果になった。ニップでは、外周部を取りにいくのを優先する事が一概に強いとは言えない事はわかった。しかし、終盤に外周部を優先して取る評価値マップの仕様であれば、作成したモンテカルロ AI は不利になる可能性が考えられる。

本研究では、モンテカルロ法と評価関数についての実験報告の段階でとどまってしまった。今後の課題と

しては、上記にも述べたのだが、まず第 1 にモンテカルロ法の改善が必要である。序盤のランダムで打つ手を評価関数や定石を組み合わせる手等に変える必要がある。ランダム要素を減らす事で無意味な打ち方などを減らす事につながると考えられる。次に、モンテカルロ法による試行回数を増やす事も必要となる。最後に、解析の最終目標は完全解析であるため、先読みが必要となってくる。しかし先読みは、深さを増やす事で処理時間が大きく増えていく。よって先読みの深さを増やすには、コンピュータの性能も必要ではあるが、並列化が必須となって重要視されてくる。並列化の実現ができれば、今後更に強い AI を作成する事が可能となってくる。

謝辞

本研究書を作成するにあたって、担当して下さった石水隆先生には、多大な時間とご迷惑をおかけしたことのお詫び及び最後までご指導して頂き、大変感謝している気持ちをこの場を借りて心より申し上げます。本当にありがとうございました。

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

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

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

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

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

[8] (2001),http://www.sze.hu/~gtakacs/download/wagnervirag_2001.pdf

[9] Jonathan Schaeffer, Neil Burch, Yngvi Bjorsson, Akihiro Kishimoto, Martin Muller,Robert Lake, Paul Lu,and Steve Suphen, Checkers is Science Vol.317, No.5844, pp1518-1522 (2007) http://www.sciencemag.org/content/317/5844/1518.full.pdf

[10] 田中哲郎,ボードゲーム「シン Janos Wagner and Istvan Virag, Solving renju, ICGA Journal, Vol.24, No.1, pp.30-35 ペイ」完全解析(修正版), http://media.itc.u-tokyo.ac.jp/ktanaka/simpei/20060307-rev.pdf

[11] Joel Feinstein, Amenor Wins World 6×6 Championships!, Forty billion noted under the tree(July 1993), pp6-8, British Fedrations newsletter,(1993), http://britishothello.org.uk/fbnall.pdf

[12] 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/

[13] 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, pp1-8(2009), http://id.nii.ac.jp/1001/00062415/

[14] 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--7(2000), http://ci.nii.ac.jp/naid/110006407446

[15] 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)

[16] 松浦美里, ニップの並列化, 近畿大学理工学部情報学科平成 23 年度卒業研究論文, 2013. http://www.info.kindai.ac.jp/~takasi-i/thesis/2012_09-1-037-0171_M_Matsuura_thesis.pdf

[17] 伊藤雄太, ニップの並列化, 近畿大学理工学部情報学科平成 23 年度卒業研究論文, 2013. http://www.info.kindai.ac.jp/~takasi-i/thesis/2012_09-1-037-0214_Y_Itoh_thesis.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モンテカルロ法を用いた AIStrategy クラスのソースを示す。

```
package net.black_cow.opennip.strategy;

import java.util.List;
import java.util.Random;

import net.black_cow.opennip.opennipcore.NipStone;
import net.black_cow.opennip.opennipcore.NipStrategy;
import net.black_cow.opennip.opennipcore.NipTable;
import net.black_cow.opennip.opennipcore.NipTableUtil;

public class AIStrategy implements NipStrategy {

	private Random rand;
	private long waitTime;

	int hyouka=0;

	/**
	 * コンストラクタ.
	 * @param waitTime 待機時間（ミリ秒）
	 */
	public AIStrategy(long waitTime) {
		rand = new Random();
		this.waitTime = waitTime;
	}

	@Override
	public int[] decide(NipTable table, NipStone stone) {

		List<int[]> list = NipTableUtil.getCanPutStoneCellList(table, stone);
		int a;
		if(table.count<20){
			a= rand.nextInt(list.size());

			try {
				Thread.sleep(waitTime);
```

```
			} catch (InterruptedException e) {
				e.printStackTrace();
			}

		}else{
		int[]valueList= playOut(table,stone);

			a=bestmove(valueList);

			try {
				Thread.sleep(waitTime);
			} catch (InterruptedException e) {
				e.printStackTrace();
			}
		}



		return list.get(a);
	}
	public  int  bestmove(int[]  valueList){//プレイアウトによる結果から評価値を決め一番高い評価となったマスを返す
		int banti=0;
		for(int i=1;i<valueList.length;i++){
			if(valueList[i]>valueList[banti]){
				banti = i;
			}
		}
		return banti;
	}



	public int[]	playOut(NipTable table, NipStone stone){//プレイアウト用のメソッド
		int trycount = 300000;
		List<int[]> list = NipTableUtil.getCanPutStoneCellList(table, stone);
		NipTable copy ;
		int[] valueList = new int[list.size()];
		for(int[] index : list) {
			int a=0;
			copy= NipTableUtil.ifYouPutStoneAt(table,index[0] ,index[1] , stone);
```

```
for(int i=0;i<trycount;i++){
	int passCount=0;
	int value=0;
	for(int j =0;2<=passCount;j++){
		List<int[]> list2;
		NipStone s;
		if(j%2 == 1){
			s =stone;
			list2  =  NipTableUtil.getCanPutStoneCellList(table,
stone);


		}else{


			if(stone.getColor().equals(NipStone.Color.WHITE)){
				s = new NipStone(NipStone.Color.BLACK);
			}else{
				s = new NipStone(NipStone.Color.WHITE);
			}
			list2=    NipTableUtil.getCanPutStoneCellList(table,
s);


		}
		if(list2.size()==0){

			passCount++;
		}else{
			int size = list2.size();
			int banti = rand.nextInt(size);
			int[] zahyou = list2.get(banti);
			int col    = zahyou[0];
			int raw = zahyou[1];
			copy.putStoneAt(col, raw, s);
			passCount = 0;
		}
	}
	int blackCount =copy.getStoneCount(NipStone.Color.BLACK);
	int whiteCount =copy.getStoneCount(NipStone.Color.WHITE);
```

```
                    if(blackCount>whiteCount){
                        if(stone.getColor().equals(NipStone.Color.WHITE)){
                            value--;
                        }else{
                            value++;
                        }
                    }else if(blackCount>whiteCount){
                        if(stone.getColor().equals(NipStone.Color.WHITE)){
                            value++;
                        }else{
                            value--;
                        }
                    }
                    valueList[a] =value;
                }
                a++;
            }
            return valueList;
        }
}
```

付録２

以下に、モンテカルロ AI と対戦に使用した評価値マップを用いた AI プログラムのソースを示す。

```
    package net.black_cow.opennip.strategy;

import java.util.ArrayList;
import java.util.List;
import java.util.Random;

import net.black_cow.opennip.opennipcore.NipStone;
import net.black_cow.opennip.opennipcore.NipStrategy;
import net.black_cow.opennip.opennipcore.NipTable;
import net.black_cow.opennip.opennipcore.NipTableUtil;

public class AIStrategy2 implements NipStrategy {
    private Random rand = new Random();
    private int[][] map = {{99 , 99 , 100, 100, 100, 100, 99 , 99 },
                                       {99 , 100, -20, -20, -20, -20, 100, 99 },
                                       {100, -20, -20,  50,  50, -20, -20, 100},
                                       {100, -20,  50,  0 ,  0 ,  50, -20, 100},
                                       {100, -20,  50,  0 ,  0 ,  50, -20, 100},
                                       {100, -20, -20,  50,  50, -20, -20, 100},
                                       {99 , 100, -20, -20, -20, -20, 100, 99 },
                                       {99 , 99 , 100, 100, 100, 100, 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 AIStrategy2(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) {

        case 0:
                value = 10;break;
        case 1:
                value = 5;break;
        case 2:
                value = 3;break;
        case 3:
                value = 1;break;
        case 4:
                value = 0;break;
        default:
                value = 0;
        }
        return value;
}
```

```
}
```