卒業研究報告書

題目

アンパンマン将棋の完全解析

指導教員

石水　隆　講師

報告者

09-1-037-0225

潘 小月

近畿大学理工学部情報学科

平成 25 年 1 月 31 日提出

概要

「アンパンマン　はじめてしょうぎ」[1](以下アンパンマン将棋とする)は、「大きくてわかりやすい駒と盤」「6 つだけの駒」「簡単だけど本格的なルール」が特長となっている子供向け将棋である。アンパンマン将棋はサイズ 3×5 の小さな将棋盤を使用し、将棋の玉将に相当する駒であるアンパンマンとバイキンマン(以下リーダーとする)を取るか、リーダーが最前線まで進めば勝ちとなる。また、本将棋と異なり、アンパンマン将棋では取った敵の駒を持ち駒にすることはできず取り捨てとなる。

アンパンマン将棋と同じく子供向け将棋である「どうぶつしょうぎ」[12]は完全解析されており、双方最善を尽くすと後手勝ちとなることが判明している[4]。一方、アンパンマン将棋は未だ完全解析されていない。そこで本研究ではアンパンマン将棋の完全解析を目指す。完全解析の前準備として、本研究ではアンパンマン将棋のプログラムを作成し、その実行結果から、アンパンマン将棋が先手有利か後手有利か引き分けかを予測する。

目次

1　序論

1.1 本研究の背景

将棋やチェス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、ゲーム終了時のプレイヤーの点数を合計すると常に 0 となる、すなわち、相手の損がそのまま自分の得となるゲームであり、一方が勝てばもう片方は必ず負けとなる。有限とは、双方のプレイヤーが可能な手の組み合わせが有限であるゲームである。ゲームに偶然の要素が無いゲームである。完全情報ゲームとは、ゲーム開始時から現在局面になるまでの全ての情報を各プレイヤーが得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。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、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[5]。チェッカーは双方最善手を指すと引き分けとな[6]。

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[7]。また、サイズ 4x4 の囲碁は双方最善手を打つと持碁(引き分け)[8]、5x5 の囲碁は黒の 25 目勝ちとなる[9]。

将棋については、盤面のサイズや使用する駒の種類を減らしたサイズ５五将棋[10]やゴロゴロ将棋[11]などのミニ将棋がある。５五将棋はサイズ 5x5 の盤と 6 種類の駒、ゴロゴロ将棋はサイズ 5x6 の盤と 4 種類の駒を使用する。図 1, 図 2 に５五将棋およびゴロゴロ将棋の盤と駒の初期配置を示す。これらは本将棋と比べて可能な局面数が少ない。しかしながら現在のところまだこれらは完全解析はされていない。

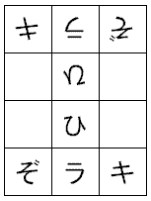
飛	角	銀	金	王
				歩
歩				
王	金	銀	角	飛

図 1　５五将棋の盤面と駒の初期配置

銀	金	王	金	銀
	歩	歩	歩	
	歩	歩	歩	
銀	金	王	金	銀

図 2 ゴロゴロ将棋の盤と駒の初期配置

完全解析されているミニ将棋として、どうぶつしょうぎ[12] (以下動物将棋とする)がある。動物将棋はサイズ 3*4 の盤と、ライオン, 象, キリン, ひよこの 4 種類の駒を使用する幼児向けのミニ将棋である。図 3 に動物将棋の盤と駒の初期配置を示す。動物将棋は完全解析により双方最善手を指した場合、78 手で後手が勝つことが判明している[4]。



ラ：ライオン
ぞ：象
キ：キリン
ひ：ひよこ

図 3 どうぶつしょうぎの盤と駒の初期配置

1.3. 完全解析されていない二人零和有限確定完全情報ゲームに対する手法

可能な局面数が多いゲームに対して完全解析を行うことは困難である。そのようなゲームに対しては確実な最適手を得ることはできないが、局面の評価値計算、定跡データベース、一定手数の先読み、終盤での必勝読みと完全読み、モンテカルロ法[19]などを用いてより有利だと思われる手を選択することができる。

定跡データベースとは、リバーシの定跡をデータベース化し、各局面で有効な定跡があればそれに従って打つという手法である。定跡データベースを使用することで強いリバーシプログラムとなる。しかし、相手があえて定跡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。

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指し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。この手法は将棋ではあまり使われないが、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[19]。

一定手数の先読みとは、可能な範囲で一定数の先の手を読み、その手から作られる局面の評価値を求め、最も評価値が高い手を採用することである。局面の評価値は、盤上に置かれている駒の種類やその位置、着手可能手の数、各駒の稼働範囲等から計算される。

ゲーム終盤になるとしそこから勝負が付くまでの手数が少なくなり、また指せる手が限定されてくるため、勝負が付くまで読み切ることが可能となる。終盤での読みは、必勝読みと完全読みがある。必勝読みとはゲーム終盤で勝敗のみを読み切り、必ず勝てる手を指すことを言う。完全読みとは、そこから得られる全ての局面を読み、最も点数の高くなる手を指すことを言う。必勝読みの方が計算時間が少なくてすむため、一般にまず必勝読みで勝ちを確定させた上で、残り手数が少なくなると完全読みに切り替えてより点数の高い勝ちを目指すことが多い。

以上の手法を用いることにより、完全解析を行わなくてもある程度の強さのプログラムを作ることが可能であり、ゲームによってはプロに勝つこともできる。

チェスでは、1997 年 5 月にチェスプログラム Deep Blue[13]が世界チャンピオン Garry Kimovich Kasparov と対戦を行い 2 勝 1 敗 3 引き分けで勝った[14]。将棋では、将棋プログラムボンクラーズ[15]が 2012 年 1 月に元プロ棋士の米長邦雄永世棋聖と対戦しボンクラーズ先手 113 手で勝った[16]。リバーシでは、リバーシプログラム Logistello[17]が 2002 年 5 月に日本チャンピオン富永健太氏と対戦を行い、Logosttelp 先手で 38 対 26 で Logostello の 12 目勝ち、富永氏先手で 23 対 41 で Logistello の 18 目勝ちであった[18]。

1.2 本研究の目的

幼児向けのミニ将棋の一つに、アンパンマンはじめてしょうぎ[1] (以下アンパンマン将棋とする)がある。アンパンマン将棋は、サイズ 3x5 の盤とアンパンマン/バイキンマン, 食パンマン/ホラーマン, カレーパンマン/ドキンちゃんの 6 個の駒を使用する。アンパンマン将棋は先手・後手で異なる名称の駒を使用するが、名称の違いのみで実質的には同一の駒と見做せるので駒の種類は 3 種類である。図 4 にアンパンマン将棋の盤と駒を示す。アンパンマン将棋の可能な局面数は、完全解析されている動物将棋と比べても少なく、充分に完全解析が可能であると考えられる。し

かし、現在のところ、アンパンマン将棋は未だ完全解析されていない。そこで本研究ではアンパンマン将棋の完全解析を目指す。本研究では、完全解析を行うための前準備として、アンパンマン将棋プログラムを作成し、アンパンマン将棋が先手有利/後手有利のどちらになるか予測する。

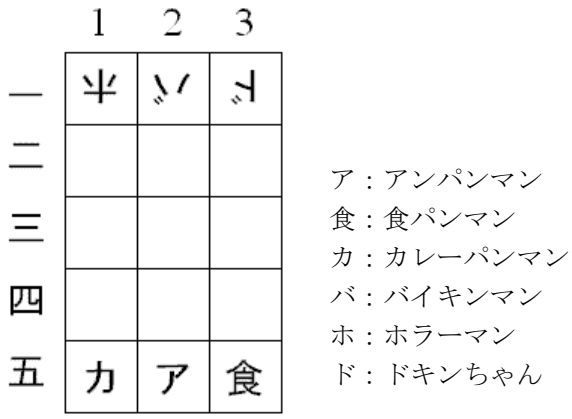


図 4 アンパンマン将棋の盤と駒の初期配置

1.3 本報告書の構成

本報告書の構成は以下のとおりである。

まず第2章において、本研究が対象とするアンパンマン将棋について説明する。続く第3章で、アンパンマン将棋の最善と思われる手を発見する手法について述べる。4 章では実験結果のまとめ、そして 5 章は結果から出た結論や今後何をする必要があるかをまとめる。

2 準備

2.1 アンパンマン将棋について

本節では、アンパンマン将棋のルールについて説明する。アンパンマン将棋では、駒は以下の６種類を使う

アンパンマン　カレーパンマン　食パンマン　バイキンマン　ドキンちゃん　ホラーマン

アンパンマン将棋では、先手をアンパンマンチーム、後手をバイキンマンチームと呼び、先手はアンパンマン, 食パンマン, カレーパンマンを、後手はバイキンマン, ホラーマン, ドキンちゃんを使用する。アンパンマンとバイキンマンは、前, 斜め前, 横の 5 方向のいずれかに 1 マス移動することができる。食パンマンとホラーマンは、前, 横の 3 方向のいずれかに 1 マス移動することができる。カレーパンマンとドキンちゃんは、前, 斜め前の 3 方向のいずれかに 1 マス移動することができる。それぞれの動きを図 5 に示す。

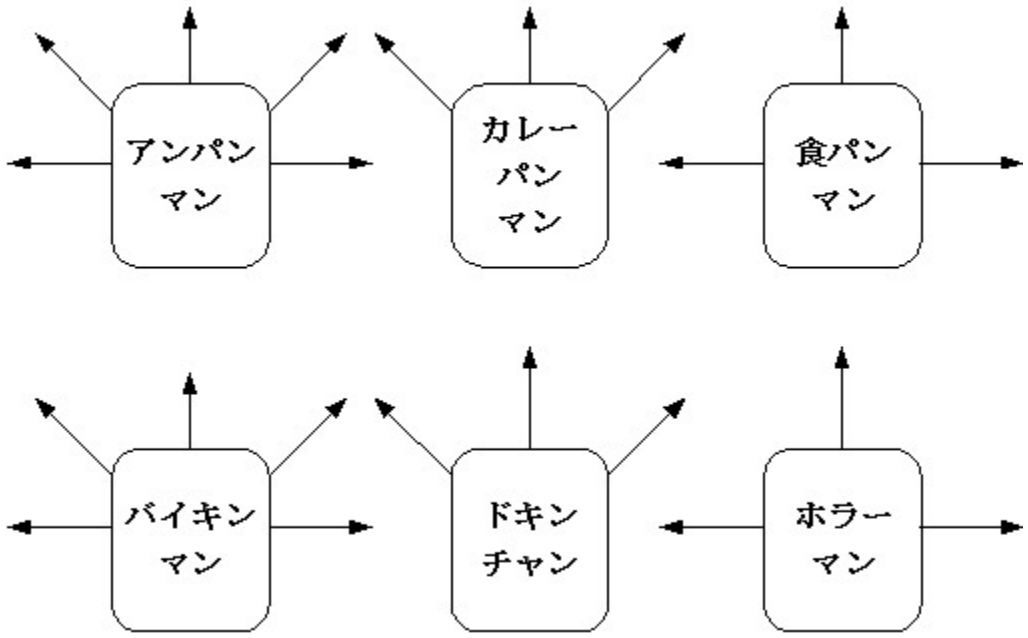


図 5　駒の動き

アンパンマンとバイキンマンは、将棋の玉将に相当する駒であり、リーダーと呼ばれる。将棋と同様に自分の駒が移動する先に相手の駒があった場合、その駒を取ることができる。ただし、アンパンマン将棋の駒は取り捨てであり、将棋のように取った駒を持ち駒にすることはできない。

アンパンマン将棋では、以下の 1、2.のいずれかを達成すると勝ちとなる。

1. 自分のリーダーがゴールする
2. 相手のリーダーを詰みにする

ゴールとはリーダーが盤上の最前列のマスに進むことである。つまり、アンパンマンは１一、２一、３一のいずれかに、バイキンマンは５一、５二、５三のいずれかに進むとゴールとなる。また、将棋と同様に、自分のリーダーに対して王手が掛かっている状態で、自分の手番でその王手を回避できる手が無い場合に詰みとなる。なお、王手とは、仮に自分の手番をパスした場合に、相手の次の手番でリーダーを取れる状態である。

また、駒の配置が同じ局面が 3 回繰り返された場合は引き分けとなる。

2.2　アンパンマン将棋の局面数

本節ではアンパンマン将棋で可能な局面数について考える。まずアンパンマンは盤上の 15 ヶ所のどこかにあるので 15 通り、バイキンマンは、アンパンマンのあるマスと、アンパンマンの前横斜め前には行けないので 11 通り、食パンマンとホラーマンはアンパンマンとバイキンマンのあるマスを除く盤上 13 ヶ所と、盤外に置けるので 14 通り、カレーパンマンとドキンちゃんは、初期配置からでは行けないマス 3 ヶ所を除く盤上 12 ヶ所と盤外に置けるので 13 通り。よって 15*11*14*14*13*13=5,465,460 通りである。

この局面数は、完全解析されている動物将棋の可能な局面数は 1,567,925,964 通り[14]と比べても充分に小さい。よって、アンパンマン将棋の完全解析を行うことは充分に可能である。

3. 研究内容

アンパンマン将棋の完全解析に先立ち、本研究ではアンパンマン将棋の AI(以下 ASAI とする)を作成した。本章では、本研究で作成した ASAI について述べる。

3.1. ASAI で用いた手法

1.3 節で述べたように、次に指す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 ASAI で用いた手法について説明する。

3.1.1. 着手可能手の決定

ある局面で可能な手は各自駒の移動可能な全ての位置に対して、その位置が空マスまたは相手駒かどうかを判定すれば発見できる。ただし自殺手を避けるため、リーダーは相手の駒が効いているマスには移動できないとする。また、自駒に王手がかかっている場合は、自分の駒の移動後に王手が解けている手以外は不可であり、王手を解除できない手は着手可能手から除く。

3.1.2. 局面の評価値計算

ある局面における評価値は、盤上にある駒の種類とその位置から計算される。一般に将棋では相手の駒を取ると有利であるので、各駒に価値を付加し、盤上にある自分の駒の価値の合計値から、相手の駒の価値の合計値を引き、その値が大きいほど有利と見ます。。駒に付加する価値は、強い駒ほど大きな値とする。アンパンマン将棋の場合は、リーダーの価値を他の駒に比べては高くする。また、リーダーが最前線に到達すると勝ちとなるので、リーダーは前進するたびにその価値を上げる。また、一般に指せ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。ただし、すでに勝負がついた局面の場合は、勝ちなら評価値無限大、負けなら評価値無限小、引き分けなら評価値 0 とする。

3.1.3. 局面の先読み

ある局面で着手可能手が複数ある場合、各手に対して、それを指した場合に局面がどう変化するかを求め、変化してできた局面の評価値を求め、評価値が最も高くなる局面を生成する手が最も有利な手と判断できる。このとき、変化したできた局面に対し、さらに着手可能手を求め、同様の操作を繰り返していくことにより正確な評価値を求めることができる。各候補手に対する評価値の計算は再帰的に行う。先読み手数が一定値に到達している場合、前節で述べた局面の評価値を候補手の評価値とする。一方、未到達の場合は、さらに次の手を先読みし、次の手の評価値の最高値を候補手の評価値とする。

3.1.4. 着手の選択

着手可能手が複数ある場合、各候補手に対して前節で述べた局面の先読みを一定手数先まで行い、評価値が最も高い手を採用する。

3.1.5. 王手の判定

王手とは、自分の駒を動かしたとき、動かした駒が次の手番で進めるマスに相手のリーダーがいて、仮に相手が手番をパスした場合に相手のリーダーが取れる手のことである。王手がかかったかどうかの判定は、駒を移動させたとき、その駒が効いている各マスを調べ、そこに相手のリーダーがいるかどうかで行える。

3.1.6. 勝敗の判定

2.1 節で述べた通り、アンパンマン将棋では、相手のリーダーを詰めるか自分のリーダーがゴールすれば勝ちとなる。詰んだかどうかは、リーダーに王手がかかっており、かつその王手を解除できる手があるかどうかで判定できる。3.1.1 節で述べたように王手が掛っている場合、王手を解除する手以外は無効である。よって、王手がかかっており、かつ有効な手が一つも無い場合は詰みと判定できる。また、ゴールしたかどうかはリーダーの Y 座標を見れば判定できる。

3.1.7. 千日手の判定

千日手（せんにちて）とは、将棋系のボードゲームにおいて駒の配置と手番が全く同じ状態が 1 局中に何回か現れること。 多くの将棋類では、それに対処するためのルールが決められている。

アンパンマン将棋では同一の局面が 3 回出てくると引き分けになる。そこで初期状態からゲーム中に現れた局面を記憶しておき、新たにできた局面と同じ駒配置の局面が過去に３回あった場合引き分けとする。先読みを行う際は、先読みで得られる局面が、以前に表れた局面と同じであるか検査し、同じであれば千日手と見做してそこから先の先読みは行わずに評価値 0 とする。

3.2. ASAI プログラム

本節では、ASAI プログラムについて述べる。付録 1 に、ASAI のソースを示す。

3.2.1. クラス Anpanman

クラス Anpanman は main メンソッドが入ってるクラスである。プログラムを開始すると、まず将棋盤を表わすクラス Board のインスタンスを生成し、先手後手の手番処理を勝敗が着くまで行う。

3.2.2. クラス Board

クラス Board は、将棋盤と盤上に置かれた駒の種類や配置を管理するクラスである。コンストラクタでは、盤上に駒が初期配置される。以下主なメソッドについて述べる。

showBoard() は将棋盤と駒を表示するメソッドである、showPiece で各プレイヤーを表示した。

player() は、先手または後手を人間が受け持つためのメソッドである。プレイヤーに動かす駒の種類と移動先の座標の入力を促し、それが有効な手であれば駒を移動させる。

com() は、先手または後手をコンピュータ受け持つためのメソッドである。3.1 節で述べた戦術に従い、最も有利と判断される手を指す。

ran() は、com()と同じく先手または後手をコンピュータが受け持つメソッドであるが、指す手をランダムに決定する。

movePiece()は駒を動かすためのメソッドであり、指定した手を指した場合に盤面がどう変化するかを計算する。

removePiece()は移動先にある駒を取るためのメソッドである。

checkwin()は勝利の判定するメソッドであり、3.1.6 節で述べた手法に従い判定する。

isChecked()はリーダーに王手がかかっているか判定するメソッドである。

isMate()は詰んだかどうかを判定するするメソッドであり、3.1.6 節で述べたように着手可能手が一つも無ければ詰みと見做す。

3.2.3. クラス Piece

クラス Piece は駒の種類と、その駒の座標およびその駒の移動可能方向を管理するメソッドである。コンストラクタでは、駒の種類に応じて駒の初期配置座標と移動可能方向をセットする。

movableList() は駒が移動可能な座標のリストを返すメソッドである。盤上の全て駒の種類と座

標が与えられたとき、各駒が移動可能な座標のリストを ArrayList<NextMove> として返す。

3.2.4. クラス NextMove

クラス NextMove は駒の移動可能な位置を表すクラスである。駒の種類と、その駒の移動先の座標を管理する。

3.3. ASAI による計算機実験

本研究では、ASAI の性能の検証および先手後手の有利さの検証のために、ASAI と、候補手からランダムで指すAI(以下RAIとする)との対戦をASAI先手、ASAI後手でそれぞれ1000回行い、勝率を計測する。ただし、ASAI の先読み手数は 3 とした。

表 1 対戦結果(先読み手数 3, 試行回数 1000 回)

	先手勝ち	後手勝ち	引き分け
ASAI 対 RAI	852	84	64
RAI 対 ASAI	56	849	95
ASAI 対 ASAI	511	323	166

4. 実験結果

本研究で作成したASAIの性能を評価するために、RAIとの対戦をASAI対RAI、RAI対ASAI、ASAI 対 ASAI でそれぞれ 1000 回行った。表 1 に対戦結果を示す。表 1 の結果より、ASAI の先読み手数が 3 程度であってもランダムに指す相手に対しては充分な性能を持っていることが示される。また、ASAI 対 ASAI の結果より、先手有利と得られる。

5. 結論・今後の課題

本研究では、アンパンマン将棋の完全解析に先立ち、アンパンマン将棋の AI (ASAI)を作成した。ASAI は、局面を一定手数先まで先読みし、先読みで得られた評価値に基いて指す手を決定する。ASAI の対戦結果から、アンパンマン将棋は先手有利と推測される。

本報告書ではアンパンマン将棋の完全解析まではできなかった。従って完全解析を行うことが今後の課題である。完全解析済みの「どうぶつしょうぎ」の可能な局面が 1,567,925,964 通りであるのに対し、アンパンマン将棋の局面は大目に見積もっても 7,138,560 通りしかない。よって完全解析を行うことは充分に可能であると予測される。完全解析を行うためには、今後全局面の勝敗について解析することを目指していく。

謝辞

本論文の作成にあたり、終始適切な助言を賜り、また丁寧に指導して下さったゼミ教員の石水隆先生に感謝致します。また、日常の議論を通じて多くの知識や示唆を頂いたり、プログラム作成の際に、多くのご指摘を下さいました情報論理工学研究室の皆様に感謝します。

参考文献

[1] アンパンマンはじめて将棋, セガトイズ (2012) http://www.segatoys.co.jp/anpan/product/popup/_legacy/learn/06.html

[2] 池 泰弘 ：コンピュータ将棋のアルゴリズム―最強アルゴリズムの探求とプログラミング, 工学社(2005)

[3] 池 泰弘 ：Java 将棋のアルゴリズム, 工学社 (2007)

[4] 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/

[5] 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

[6] 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

[7] 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

[8] 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/

[9] 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).

[10] 日本 5 五将棋連盟, http://www.geocities.co.jp/Playtown-Spade/8662/

[11] 「ごろごろどうぶつしょうぎ」発売開始！, お知らせ, 日本将棋連盟, 2012 年 11 月 26 日, (2012), http://www.shogi.or.jp/topics/2012/11/post-652.html

[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] IBM100 – Deep Blue, IBM, (1997), http://www-03.ibm.com/ibm/history/ibm100/us/en/icons/deepblue/

[14] Michael Khodarkovsky and Leonid Shamkvoich, 人間対機械 ― チェス世界チャンピオンとスーパーコンピューターの闘いの記録, 毎日コミュニケーションズ, (1998)

[15] 伊藤英紀, A 級リーグ差し手 1 号, (2013), http://aleag.cocolog-nifty.com/

[16] 米長邦雄, われ敗れたり コンピュータ棋戦のすべてを語る, 中央公論社, (2012).

[17] Michael Buro , LOGISTELLO, 2002, https://skatgame.net/mburo/log.html

[18] Michael Buro , Tominaga vs. Logistello, 2002, https://skatgame.net/mburo/iwec.html

[19] 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).

付録

以下に本研究で作成したアンパンマン将棋プログラムのソースを示す。

「Anpanman ソース」

```
package anpan;

import java.util.Scanner;
import java.util.ArrayList;
import java.io.*;

public class Anpanman {
	final static int ANPANMAN = 1;     // アンパンマン
	final static int SHOKUPANMAN = 2;  // 食パンマン
	final static int CURRYPANMAN = 3;  // カレーパンマン
	final static int BAIKINMAN = -1;   // バイキンマン
	final static int HORRORMAN = -2;   // ホラーマン
	final static int DOKINCHAN = -3;   // ドキンちゃん
	final static int EMPTY = 0;        // 空白
	final static int BORDER = Integer.MAX_VALUE; // 盤外外
	static FileWriter pass, rPass;     // 棋譜出力用
	static PrintWriter fp, rfp;

	public static void main (String[] args) {

		boolean isCom[] = {false, false};
		Scanner keyBoardScanner = new Scanner(System.in);
		String input; // 入力用;
		String score; // 棋譜;
		FileWriter pass = null; // 棋譜出力用ファイル
		PrintWriter fp = null;  // 棋譜出力用ファイルのポインタ
		int win=0, lose=0, draw=0; // 勝敗数

		try {
			pass = new FileWriter ("anpanmanWinner.txt");
			fp = new PrintWriter (pass);
		} catch (IOException e) {
			System.err.println (e);
		}

		int roop=0, depth=0;
		System.out.println ("ループ回数を入れてください");
		do {
			roop = keyBoardScanner.nextInt();
		} while (roop <= 0);
```

```
System.out.print ("アンパンマンチームはASAIが持ちますか？ (Y/N) ");
input = keyBoardScanner.next();
if (input.equals ("Y") || input.equals ("y")) {
        isCom[0] = true;
}
System.out.print ("バイキンマンチームはASAIが持ちますか？ (Y/N) ");
input = keyBoardScanner.next();
if (input.equals ("Y") || input.equals ("y")) {
        isCom[1] = true;
}
if (isCom[0] || isCom[1]) {
        System.out.print ("ASAIの先読み数はいくらにしますか？");
        do {
                depth = keyBoardScanner.nextInt();
        } while (roop < 0);
}

for (int r=0; r<roop; ++r) {
        Board board = new Board();
        board.resetMoves();
        board.setMaxDepth (depth);
        while (true) {
                board.createMovableList (0);
                        // アンパンマンチームが移動可能な手を求める
                if (board.checkWin (1)) break;
                if (isCom[0]) {
                        score = board.com (0); // ASAIが指す
                } else {
                        score = board.ran (0); // RAIが指す
                }
                board.createMovableList (1);
                        // バイキンマンチームが移動可能な手を求める
                if (board.checkWin (0)) break;
                if (isCom[1]) {
                        score = board.com (1); // ASAIが指す
                } else {
                        score = board.ran (1); // RAIが指す
                }
        }
        int winner = board.winner();
        if (winner == 1) {          // 先手勝ち
                ++win;
                System.out.println (r + ": 先手勝ち (" + win + ":"
                        + lose + ":" + draw + ")");
                fp.println (r + ": 先手勝ち (" + win + ":" + lose + ":" + draw + ")");
        } else if (winner == -1) { // 後手勝ち
```

```
                                ++lose;
                                System.out.println (r + ": 後手勝ち (" + win + ":"
                                        + lose + ":" + draw + ")");
                                fp.println (r + ": 後手勝ち (" + win + ":" + lose + ":" + draw + ")");
                        } else {                        // 引き分け
                                ++draw;
                                System.out.println (r + ": 引き分け (" + win + ":"
                                        + lose + ":" + draw + ")");
                                fp.println (r + ": 引き分け (" + win + ":" + lose + ":" + draw + ")");
                        }
                }
                System.out.println ("(先手勝ち:後手勝ち:引き分け) = (" + win + ":" + lose
                                + ":" + draw + ")");
                fp.println ("(先手勝ち:後手勝ち:引き分け) = (" + win + ":" + lose + ":" + draw + ")");
                fp.close();
        }
}
```

「Board ソース」

```
package anpan;

import java.util.Scanner;
import java.util.ArrayList;
import java.io.*;

public class Anpan {
        final static int ANPANMAN = 1;      // アンパンマン
        final static int SHOKUPANMAN = 2;  // 食パンマン
        final static int CURRYPANMAN = 3;  // カレーパンマン
        final static int BAIKINMAN = -1;    // バイキンマン
        final static int HORRORMAN = -2;    // ホラーマン
        final static int DOKINCHAN = -3;    // ドキンちゃん
        final static int EMPTY = 0;         // 空白
        final static int BORDER = Integer.MAX_VALUE; // 盤外外
        static FileWriter pass, rPass;      // 棋譜出力用
        static PrintWriter fp, rfp;


        public static void main (String[] args) {
                Board board = new Board();
                boolean isCom[] = {false, false};
                Scanner keyBoardScanner = new Scanner(System.in);
                String input; // 入力用;
                String score; // 棋譜;
                FileWriter pass = null; // 棋譜出力用ファイル
```

```
                PrintWriter fp = null;  // 棋譜出力用ファイルのポインタ

                try {
                        pass = new FileWriter ("anpanmanScore.txt");
                        fp = new PrintWriter (pass);
                } catch (IOException e) {
                        System.err.println (e);
                }

                System.out.print ("アンパンマンチームはCOMが持ちますか？ (Y/N) ");
                input = keyBoardScanner.next();
                if (input.equals ("Y") || input.equals ("y")) {
                        isCom[0] = true;
                }
                System.out.print ("バイキンマンチームはCOMが持ちますか？ (Y/N) ");
                input = keyBoardScanner.next();
                if (input.equals ("Y") || input.equals ("y")) {
                        isCom[1] = true;
                }
                while (true) {
                        board.showBoard();
                        board.createMovableList (0); // アンパンマンチームが移動可能な手を求める
                        //System.out.println (board.value(1));
                        if (board.isChecked (0)) System.out.println ("王手！");
                        if (board.checkWin (1)) break;
                        if (isCom[0]) {
                                score = board.com (0);
                        } else {
                                score = board.player (0);
                        }
                        fp.print (score); // 棋譜出力
                        board.showBoard();
                        board.createMovableList (1); // バイキンマンチームが移動可能な手を求める
                        //System.out.println (board.value(0));
                        if (board.isChecked (1)) System.out.println ("王手！");
                        if (board.checkWin (0)) break;
                        if (isCom[1]) {
                                score = board.com (1);
                        } else {
                                score = board.player (1);
                        }
                        fp.println (score); // 棋譜出力
                }
                fp.close();
        }
}
```

「NextMove ソース」

```
package anpan;

/**
 * 駒の移動可能な位置を表すクラス
 */
public class NextMove {
	final static int ANPANMAN = 1;     // アンパンマン
	final static int SHOKUPANMAN = 2;  // 食パンマン
	final static int CURRYPANMAN = 3;  // カレーパンマン
	final static int BAIKINMAN = -1;   // バイキンマン
	final static int HORRORMAN = -2;   // ホラーマン
	final static int DOKINCHAN = -3;   // ドキンちゃん

	final static boolean isChessStyleScore = true; // 棋譜表記をチェス式か将棋式か？

	int type;      // 駒の種類
	int nextFile;  // 移動先の X 座標
	int nextRank;  // 移動先の Y 座標
	int value;     // 移動した場合の盤面の評価値

	/**
	 * コンストラクタ
	 * @param int type 駒の種類
	 * @param int nextFile 移動先の X 座標
	 * @param int nextRank 移動先の Y 座標
	 */
	public NextMove (int type, int nextFile, int nextRank) {
		this.type = type;
		this.nextFile = nextFile;
		this.nextRank = nextRank;
	}

	/**
	 * 駒の種類を返す
	 * @return 駒の種類
	 */
	public int type() {
		return type;
	}

	/**
	 * 移動先の X 座標を返す
	 * @return X 座標
	 */
	public int nextFile() {
```

```
        return nextFile;
}

/**
 * 移動先の Y 座標を返す
 * @return Y 座標
 */
public int nextRank() {
        return nextRank;
}


/**
 * 移動した場合の盤面の評価値を返す
 * @return 評価値
 */
public int value() {
        return value;
}

/**
 * 評価値をセットする
 * @param value 評価値
 */
public void setValue (int value) {
        this.value = value;
}

/**
 * 棋譜を返す
 * 変数 isChessStyle によりチェス風の棋譜あるいは将棋風の棋譜か変化する
 * @return 棋譜の文字列表現
 */
public String toString () {
        String score; // 棋譜
        if (isChessStyleScore) { // チェス風の棋譜を作成
                switch (type) {
                case ANPANMAN :    score = "A"; break;
                case SHOKUPANMAN : score = "S"; break;
                case CURRYPANMAN : score = "C"; break;
                case BAIKINMAN :   score = "B"; break;
                case HORRORMAN :   score = "H"; break;
                case DOKINCHAN :   score = "D"; break;
                default :          score = "?"; break;
                }
                switch (nextFile) {
                case 1 :  score += "a"; break;
```

```
                    case 2 :  score += "b"; break;
                    case 3 :  score += "c"; break;
                    default : score += "?"; break;
                    }
                    switch (nextRank) {
                    case 1 :  score += "1 "; break;
                    case 2 :  score += "2 "; break;
                    case 3 :  score += "3 "; break;
                    case 4 :  score += "4 "; break;
                    case 5 :  score += "5 "; break;
                    default : score += "? " ; break;
                    }
                } else { // 将棋風の棋譜を作成
                    switch (nextFile) {
                    case 1 :  score = "１"; break;
                    case 2 :  score = "２"; break;
                    case 3 :  score = "３"; break;
                    default : score = "？"; break;
                    }
                    switch (nextRank) {
                    case 1 :  score += "一"; break;
                    case 2 :  score += "二"; break;
                    case 3 :  score += "三"; break;
                    case 4 :  score += "四"; break;
                    case 5 :  score += "五"; break;
                    default : score += "？" ; break;
                    }
                    switch (type) {
                    case ANPANMAN :    score += "あ  "; break;
                    case SHOKUPANMAN : score += "食  " ; break;
                    case CURRYPANMAN : score += "カ  "; break;
                    case BAIKINMAN :   score += "バ  "; break;
                    case HORRORMAN :   score += "ホ  "; break;
                    case DOKINCHAN :   score += "ド  "; break;
                    default :          score += "？  "; break;
                    }
                }
                return score;
        }
}
```

「Piece ソース」

```
package anpan;

import java.util.ArrayList;
```

```
public class Piece {
        final static int ANPANMAN = 1;     // アンパンマン
        final static int SHOKUPANMAN = 2;  // 食パンマン
        final static int CURRYPANMAN = 3;  // カレーパンマン
        final static int BAIKINMAN = -1;   // バイキンマン
        final static int HORRORMAN = -2;   // ホラーマン
        final static int DOKINCHAN = -3;   // ドキンちゃん
        final static int EMPTY = 0;        // 空白
        final static int BORDER = Integer.MAX_VALUE; // 盤外

        final static int[] ANPANMAN_MOVABLE_FILE_VECTOR    = {-1,-1, 0, 1, 1};
                                                           // アンパンマンの移動可能X方向
        final static int[] ANPANMAN_MOVABLE_RANK_VECTOR    = { 0,-1,-1,-1, 0};
                                                           // アンパンマンの移動可能Y方向
        final static int[] SHOKUPANMAN_MOVABLE_FILE_VECTOR = {-1, 0, 1};
                                                           // 食パンマンの移動可能X方向
        final static int[] SHOKUPANMAN_MOVABLE_RANK_VECTOR = { 0,-1, 0};
                                                           // 食パンマンの移動可能Y方向
        final static int[] CURRYPANMAN_MOVABLE_FILE_VECTOR = {-1, 0, 1};
                                                           // カレーパンマンの移動可能X方向
        final static int[] CURRYPANMAN_MOVABLE_RANK_VECTOR = {-1,-1,-1};
                                                           // カレーパンマンの移動可能Y方向
        final static int[] BAIKINMAN_MOVABLE_FILE_VECTOR   = {-1,-1, 0, 1, 1};
                                                           // バイキンマンの移動可能X方向
        final static int[] BAIKINMAN_MOVABLE_RANK_VECTOR   = { 0, 1, 1, 1, 0};
                                                           // バイキンマンの移動可能Y方向
        final static int[] HORRORMAN_MOVABLE_FILE_VECTOR   = {-1, 0, 1};
                                                           // ホラーマンの移動可能X方向
        final static int[] HORRORMAN_MOVABLE_RANK_VECTOR   = { 0, 1, 0};
                                                           // ホラーマンの移動可能Y方向
        final static int[] DOKINCHAN_MOVABLE_FILE_VECTOR   = {-1, 0, 1};
                                                           // ドキンちゃんの移動可能X方向
        final static int[] DOKINCHAN_MOVABLE_RANK_VECTOR   = { 1, 1, 1};
                                                           // ドキンちゃんの移動可能Y方向

        boolean isOnBoard; // 盤上に駒があるか
        int file;          // 駒のX座標
        int rank;          // 駒のY座標
        int type;          // 駒の種類
        int movableRankVector[], movableFileVector[];
                                   // 移動可能方向(駒の現在位置を(0,0)とした場合の相対位置)

        /**
         * コンストラクタ
         * 駒を生成し初期位置に置く
         * @param int type 駒の種類
```

```
 */
public Piece (int type) {
         this.type = type;
         setMovableVector();
         setInitialPosition();
}

/**
 * コンストラクタ
 * 駒を生成し指定した位置に置く
 * @param int type 駒の種類
 * @param int
 */
public Piece (int type, int file, int rank) {
         this.type = type;
         setMovableVector();
         this.file = file;
         this.rank = rank;
         this.isOnBoard = true;
}

/**
 * 駒の移動可能方向をセット
 */
private void setMovableVector() {
         switch (type) { // 駒の種類により分岐
         case ANPANMAN : // アンパンマンの場合
                  movableFileVector = ANPANMAN_MOVABLE_FILE_VECTOR;
                  movableRankVector = ANPANMAN_MOVABLE_RANK_VECTOR;
                  break;
         case SHOKUPANMAN : // 食パンマンの場合
                  movableFileVector = SHOKUPANMAN_MOVABLE_FILE_VECTOR;
                  movableRankVector = SHOKUPANMAN_MOVABLE_RANK_VECTOR;
                  break;
         case CURRYPANMAN : // カレーパンマンの場合
                  movableFileVector = CURRYPANMAN_MOVABLE_FILE_VECTOR;
                  movableRankVector = CURRYPANMAN_MOVABLE_RANK_VECTOR;
                  break;
         case BAIKINMAN : // バイキンマンの場合
                  movableFileVector = BAIKINMAN_MOVABLE_FILE_VECTOR;
                  movableRankVector = BAIKINMAN_MOVABLE_RANK_VECTOR;
                  break;
         case HORRORMAN : // ホラーマンの場合
                  movableFileVector = HORRORMAN_MOVABLE_FILE_VECTOR;
                  movableRankVector = HORRORMAN_MOVABLE_RANK_VECTOR;
                  break;
         case DOKINCHAN : // ドキンちゃんの場合
```

```
                movableFileVector = DOKINCHAN_MOVABLE_FILE_VECTOR;
                movableRankVector = DOKINCHAN_MOVABLE_RANK_VECTOR;
                break;
        default : // それ以外の場合
                System.out.println ("駒の種類を認識できません");
        }
}

/**
 * 駒を盤上の初期位置にセット
 */
private void setInitialPosition(){
        switch (type) {
        case ANPANMAN :    // アンパンマンの場合
                rank = 5;
                file = 2;
                break;
        case SHOKUPANMAN : // 食パンマンの場合
                rank = 5;
                file = 3;
                break;
        case CURRYPANMAN : // カレーパンマンの場合
                rank = 5;
                file = 1;
                break;
        case BAIKINMAN :   // バイキンマンの場合
                rank = 1;
                file = 2;
                break;
        case HORRORMAN :   // ホラーマンの場合
                rank = 1;
                file = 1;
                break;
        case DOKINCHAN :   // ドキンちゃんの場合
                rank = 1;
                file = 3;
                break;
        }
        isOnBoard = true;
}

/**
 * 駒を盤上の指定した座標にセット
 * @param int file X座標
 * @param int rank Y座標
 */
private void setPosition (int file, int rank){
```

```
        this.file = file;
        this.rank = rank;
        isOnBoard = true;
}

/**
 * 駒を盤上に置く
 */
public void setOnBoard() {
        isOnBoard = true;
}

/**
 * 駒を盤上から削除
 */
public void removeFromBoard() {
        isOnBoard = false;
}

/**
 * 盤上に駒が存在するか
 * @return 駒が存在するか
 */
public boolean isOnBoard() {
        return isOnBoard;
}

/**
 * 指定した座標に駒が存在するか
 * @param int file x座標
 * @param int rank y座標
 * @return 駒が存在するか
 */
public boolean isThere (int file, int rank) {
        if (isOnBoard) {
                return ((this.file == file) && (this.rank == rank));
        } else return false;
}

/**
 * x座標を返す
 * @return x座標
 */
public int file(){
        return this.file;
}
```

```
/**
 * y 座標を返す
 * @return y 座標
 */
public int rank(){
        return this.rank;
}

/**
 * 駒の種類を返す
 * @return 駒の種類
 */
public int type(){
        return this.type;
}

/**
 * 駒名を返す
 * @return 駒名
 */
public String name() {
        switch (type) {
        case ANPANMAN :
                return "アンパンマン";
        case SHOKUPANMAN :
                return "食パンマン";
        case CURRYPANMAN :
                return "カレーパンマン";
        case BAIKINMAN :
                return "バイキンマン";
        case HORRORMAN :
                return "ホラーマン";
        case DOKINCHAN :
                return "ドキンちゃん";
        default :
                return "？";
        }
}

/**
 * 駒を指定した座標に移動する
 * @param nextFile 移動する X 座標
 * @param nextRank 移動する Y 座標
 */
public void move (int nextFile,int nextRank) {
        file = nextFile;
        rank = nextRank;
```

```
}

/**
 * 駒が指定した座標に移動できるか
 * 他の駒は無視して自分が移動できるかのみを判断する
 * @param int nextFile 移動したいX座標
 * @param int nextRank 移動したいY座標
 * @return 移動できるか
 */
public boolean movable (int nextFile,int nextRank) {
        if (!isOnBoard) // すでに取られている場合
                return false;
        if (nextFile < 1 || nextFile > 3 || nextRank < 1 || nextRank > 5)
                                                        // 盤外を指定した場合
                return false;
        for (int i = 0; i < movableFileVector.length; ++i) {
                if ((nextFile == file + movableFileVector[i])
                && (nextRank == rank + movableRankVector[i]))
                        return true;
        }
        return false;
}

/**
 * 駒が指定した座標に移動できるか
 * 他の駒も考慮して移動できるかを判断する
 * @param int[][] board 現在の盤
 * @param int nextFile 移動したいX座標
 * @param int nextRank 移動したいY座標
 * @return 移動できるか
 */
public boolean movable (int[][] board, int nextFile,int nextRank) {
        if (!isOnBoard) // すでに取られている場合
                return false;
        ArrayList<NextMove> movableList = movableList (board);
        for (int i = 0; i < movableList.size(); ++i) {
                if ((movableList.get(i).nextFile() == nextFile)
                && (movableList.get(i).nextRank() == nextRank)) {
                        return true;
                }
        }
        return false;
}

/**
 * 駒を指定した座標に移動する
 * 他の駒は無視して自分が移動できるかのみを判断し、可能なら移動する
```

```
 * @param int nextFile 移動したいX座標
 * @param int nextRank 移動したいY座標
 * @return 移動できたか
 */
public boolean checkAndMove (int nextFile,int nextRank) {
        if (movable (nextFile, nextRank)) {
                file = nextFile;
                rank = nextRank;
                return true;
        } else return false;
}

/**
 * 駒を指定した座標に移動する
 * 他の駒の位置も考慮して移動できるか判断し、可能なら移動する
 * @param int[][] board 現在の盤
 * @param int nextFile 移動したい x座標
 * @param int nextRank 移動したい y座標
 * @return 移動できたか
 */
public boolean checkAndMove (int[][] board, int nextFile,int nextRank) {
        if (movable (board, nextFile, nextRank)) {
                file = nextFile;
                rank = nextRank;
                return true;
        } else return false;
}

/**
 * 移動可能な座標のリストを返す
 * @param int[][] board 現在の盤
 * @return 移動可能な座標のリスト
 */
public ArrayList<NextMove> movableList (int[][] board) {
        ArrayList<NextMove> movableList = new ArrayList<NextMove>();
        boolean[][] isMovable = {{false, false, false, false, false}, // 移動可能な位置
                        {false, true,  true,  true,  false},
                        {false, true,  true,  true,  false},
                        {false, true,  true,  true,  false},
                        {false, true,  true,  true,  false},
                        {false, true,  true,  true,  false},
                        {false, false, false, false, false}};

        switch (type) { // 他の駒による移動不可能な位置の判定
        case ANPANMAN : // アンパンマンの場合
                for (int r = 1; r <= 5; ++r) {
                        for (int f = 1; f <=3; ++f) {
```

```
                        switch (board[r][f]) {
                        case ANPANMAN :  // 自分の駒がある位置へは移動不可
                        case SHOKUPANMAN :
                        case CURRYPANMAN :
                                isMovable[r][f] = false;
                                break;
                        case BAIKINMAN : // バイキンマンに取られる位置へは移動不可
                                isMovable[r][f-1] = false;
                                isMovable[r+1][f-1] = false;
                                isMovable[r+1][f] = false;
                                isMovable[r+1][f+1] = false;
                                isMovable[r][f+1] = false;
                                break;
                        case HORRORMAN : // ホラーマンに取られる位置へは移動不可
                                isMovable[r][f-1] = false;
                                isMovable[r+1][f] = false;
                                isMovable[r][f+1] = false;
                                break;
                        case DOKINCHAN : // ドキンちゃんに取られる位置へは移動不可
                                isMovable[r+1][f-1] = false;
                                isMovable[r+1][f] = false;
                                isMovable[r+1][f+1] = false;
                                break;
                        }
                }
        }
case SHOKUPANMAN : // 食パンマンの場合
case CURRYPANMAN : // カレーパンマンの場合
        for (int r = 1; r <= 5; ++r) {
                for (int f = 1; f <=3; ++f) {
                        switch (board[r][f]) {
                        case ANPANMAN :  // 自分の駒がある位置へは移動不可
                        case SHOKUPANMAN :
                        case CURRYPANMAN :
                                isMovable[r][f] = false;
                                break;
                        }
                }
        }
        break;
                case BAIKINMAN : // バイキンマンの場合
        for (int r = 1; r <= 5; ++r) {
                for (int f = 1; f <=3; ++f) {
                        switch (board[r][f]) {
                        case BAIKINMAN : // 自分の駒がある位置へは移動不可
                        case HORRORMAN :
                        case DOKINCHAN :
```

```
                                        isMovable[r][f] = false;
                                        break;
                                case ANPANMAN : // アンパンマンに取られる位置へは移動不可
                                        isMovable[r][f-1] = false;
                                        isMovable[r-1][f-1] = false;
                                        isMovable[r-1][f] = false;
                                        isMovable[r-1][f+1] = false;
                                        isMovable[r][f+1] = false;
                                        break;
                                case SHOKUPANMAN : // 食パンマンに取られる位置へは移動不可
                                        isMovable[r][f-1] = false;
                                        isMovable[r-1][f] = false;
                                        isMovable[r][f+1] = false;
                                        break;
                                case CURRYPANMAN : // カレーパンマンに取られる位置へは移動不可
                                        isMovable[r-1][f-1] = false;
                                        isMovable[r-1][f] = false;
                                        isMovable[r-1][f+1] = false;
                                        break;
                                }
                        }
                }
        case HORRORMAN : // ホラーマンの場合
        case DOKINCHAN : // ドキンちゃんの場合
                for (int r = 1; r <= 5; ++r) {
                        for (int f = 1; f <=3; ++f) {
                                switch (board[r][f]) {
                                case BAIKINMAN :  // 自分の駒がある位置へは移動不可
                                case HORRORMAN :
                                case DOKINCHAN :
                                        isMovable[r][f] = false;
                                        break;
                                }
                        }
                }
                break;
        }
        for (int i = 0; i < movableFileVector.length; ++i) { // 移動可能かチェック
                int nextFile = file + movableFileVector[i];
                int nextRank = rank + movableRankVector[i];
                if (isMovable [nextRank][nextFile]) {
                        NextMove nextMove = new NextMove (type, nextFile, nextRank);
                                                        // 移動可能リストに挿入
                        movableList.add (nextMove);
                }
        }
        return movableList;
```

```
	}

	/**
	 * 移動可能な座標を表示する
	 * @param int[][] board 現在の盤
	 */
	public void showMovable (int[][] board) {
		ArrayList<NextMove> movableList = movableList (board); // 移動可能な座標の判定

		if (movableList.size() == 0) { // 駒が移動可能な位置が無い場合
			System.out.println (name() + "が進める位置はありません");
		} else {
			System.out.print (name() + "は現在(" + file + "," + rank + ")にいて");
			for (int i = 0; i < movableList.size(); ++i) {
				int nextFile = movableList.get(i).nextFile();
				int nextRank = movableList.get(i).nextRank();
				System.out.print ("(" +  nextFile + "," + nextRank + ")");
			}
			System.out.println ("へ移動できます");
		}
	}

	/**
	 * クローン生成
	 */
	public Piece clone() {
		Piece newPiece = new Piece (this.type, this.file, this.rank);
		if (!isOnBoard) newPiece.removeFromBoard();
		return newPiece;
	}
}
```