卒業研究報告書

題目

アンパンマンしょうぎの完全解析

指導教員

石水　隆　講師

報告者

08-1-037-0196

西川　千晶

近畿大学理工学部情報学科

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2012年に女流棋士の北尾まどか初段によって考案されたボードゲームである。将棋に類似しているが、将棋と比べて非常に簡潔なルールになっている。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。

アンパンマン将棋は二人完全情報零和ゲームであり、すべての局面の理論値（勝ち、負け、引き分けのいずれか）が決定可能である。従って、理論上は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。多くのボードゲームでは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が、アンパンマン将棋では可能な局面の総数は7,138,560通りと小さいため、完全解析を行うことは充分に可能だと考えられる。

本報告では完全解析に先立ちアンパンマン将棋プログラムを Java を用いて作成した。本研究で作成したプログラムは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、王手を回避できる手が存在しない場合は詰みとなる。

また評価値は、盤上に存在する駒の種類とその位置により決定される。一般に将棋は自駒が多いと有利、相手駒が多いと不利であるので、自駒に正の価値、相手駒に負の価値を付け、その合計値を評価値の基準のひとつとして用いる。リーダーは最前列まで進むと勝ちであるので、前進するにつれてその価値を大きくする。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。

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、前述の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。優勢であれば千日手を避けるようにし、劣勢であれば、千日手を積極的に行い引き分けに持っていくように作成した。ASAI の性能を評価するために、ASAI と RAI との対戦を ASAI 先手、RAI 先手それぞれで1000回行った。ASAI の先読み手数が6手の場合、対戦成績は ASAI 先手で969勝3負28引き分け、RAI 先手で19勝935負46引き分けであった。また、ASAI 同士の対戦を1000回行った時、ASAI 先手で534勝225負241引き分けとなった。この結果よりアンパンマン将棋は先手優勢であることが分かった。

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、プレーしている全プレーヤーの利得の合計が常にゼロ、または個々のプレーヤーの指す手の組み合わせに対する利得の合計が全て一定の数値（零和）となるゲームのことである。有限とは、そのゲームにおける各プレーヤーの可能な手の組み合わせの総数が有限であるゲームのことである。確定とはプレーヤーの着手以外にゲームに影響を与える偶然の要素が入り込まないという意味である。完全情報ゲームとは、各プレイヤーが自分の手番において、これまでの各プレイヤーの行った選択（あるいは意思決定）について全ての情報を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。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。現在、将棋やチェスなどのボードゲームで人間に負けないプログラムが話題になっている事例がいくつかあり[14][16][19]、ゲームによってはその道のプロに勝てるものもある。

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[5]。チェッカーは双方最善手を指すと引き分けとなる[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	玉
				歩
歩				
玉	金	銀	角	飛

図1 5五将棋の盤面と駒の初期配置

銀	金	玉	金	銀
	歩	歩	歩	
	歩	歩	歩	
銀	金	玉	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3x4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[3]。

キ	ラ	ぞ
	ひ	
	ひ	
ぞ	ラ	キ

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

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, 対戦データベース、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、定跡というお互いが最善と考えられる手を行った場合の一連の手を多数データベースとして持っておき、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将棋プログラムとなる。しかし、相手があえて定跡以外の手を指すなどして、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。また、歴史の浅いゲームでは定跡が確立されていないことも多い。

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将棋プログラムになる。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指されたりした場合には使えないという欠点がある。

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。

モンテカルロ法とは、乱数を用いたシミュレーションを何度も行うことにより近似解を求める計算手法である。解析的に解くことができない問題でも、十分多くの回数シミュレーションを繰り返すことにより、近似的に解を求めることができる。適用範囲が広く、問題によっては他の数値計算手法より簡単に適用できるが、高い制度を得ようとすれば計算回数が膨大になってしまうという弱点もある。この手法は将棋プログラム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[17]。

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ロジステロ[18]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、ロジステロ先手で38対26でロジステロの12目勝ち、富永氏先手で23対41でロジステロの18目勝ちであった[19]。

1.4 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女流棋士の北尾まどか初段によって考案されたボードゲームである。ゲーム名から分かるようにルールは将棋に類似しているが、児童への普及を主目的としているため以下のような簡潔なルールになっている。

アンパンマン将棋ではサイズ3x5の将棋盤を用い、アンパンマン, 食パンマン, カレーパンマン, バイキンマン, ホラーマン, ドキンちゃんの6個の駒を用いる。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将棋盤と駒の初期配置を示す。五段目(1五, 2五, 3五)、一段目(1一, 2一, 3一)をそれぞれ(先手、後手)の陣地と呼ぶ。

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。

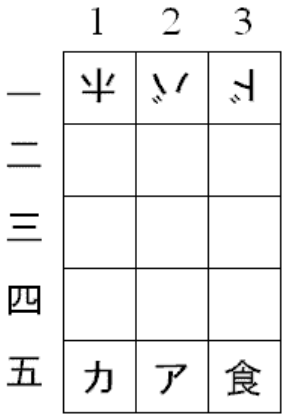

図4 アンパンマン将棋の将棋盤と駒の初期配置

1.5 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章で実験結果と考察を述べ、5章で結論と今後の課題を述べる。

2 アンパンマン将棋

本章では、アンパンマン将棋のルールについて説明する。

2.1 アンパンマン将棋の駒

アンパンマン将棋では、駒は以下の6個を使用する。

アンパンマンとバイキンマン（以下リーダーとする）は、将棋の玉将に相当する駒であり、前, 斜め前, 両横の5方向のいずれかに1マス動ける。カレーパンマンとドキンちゃんは、前と斜めの3方向のいずれかに1マス動ける。食パンマンとホラーマンは、前と両横の3方向のいずれかに1マス動ける。

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、将棋の玉将に相当する駒であり、リーダーと呼ばれる。アンパンマン将棋の駒を図5に示す。各駒には、アンパンマンのキャラクターと、その駒が動ける方向を示す矢印が描かれている。アンパンマン将棋には駒の成りは無いため、駒の裏面は空白である。

図5 アンパンマン将棋の駒

2.2 アンパンマン将棋の進行と勝敗

アンパンマン将棋は、将棋と同様にプレイヤー2人で遊ぶゲームであり、初期局面から交互に1手ずつ駒を動かす。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、その駒を捕まえることができる。捕まえた敵の駒は盤上から取り除かれる。従って、アンパンマン将棋の駒はチェスと同じく取り捨てであり、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、相手のリーダーを捕まえるか、相手の陣地に自分のリーダーが入ると勝ちとなる。リーダーを捕まえるとは、将棋で言えば相手の玉将を詰みにすることである。すなわち、相手のリーダーに対して王手が掛かっている状態で、相手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。相手の陣地にリーダーが入るとは、リーダーが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、駒の配置が同じ局面が3回出てきた場合は千日手で引き分けとなる。

2.3 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。
アンパンマン将棋の可能な局面数は、アンパンマンは盤外には置けないため15通り、バイキンマンは盤外とアンパンマンがある位置、アンパンマンの隣・前・ななめ前には置けないため11通り、食パンマンとホラーマンは盤外を含めどこでも置けるので各16通り、カレーパンマンとドキンちゃんは初期配置から到達できないマスが3箇所あるため各13通り。それを全て掛けると総局面の見積もりは7,138,560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[3]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まずアンパンマン将棋プログラムを Java を用いて作成した。本研究で作成したプログラムは対人戦と対 AI 戦を行うことができる。
対 AI 戦で用いられるアンパンマン将棋の AI（以下 ASAI とする）はある局面で指せる手が複数ある場合、数手先の局面を先読みし各局面の評価値を計算し、最も評価値の高い手を採用する。

3.1 ASAI で用いた手法

本節では ASAI で用いた手法について説明する。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。
モンテカルロ法はランダムで良い手を検索する手法であるので、今回は、完全解析が目的であり、ASAIは完全解析を行うための前準備として行うものである。従って、ランダムに手を検索するモンテカルロ法は今回の目的にはそぐわない。また、アンパンマン将棋は発売直後であるため、定跡はまだ確立されていないため、定跡データベースは、使用できない。また、リバーシ等と異なり将棋では何手で勝負が付くかはっきりしないため、何処から終盤に入るのかを判定するのは難しい。よって終盤での必勝読みは、今回は使用しなかった。

3.1.1 候補手の発見

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、王手を回避できる手が存在しない場合は詰みとなる。

3.1.2 評価値の計算

ある局面の評価値は、盤上に存在する駒の種類とその位置により決定される。一般に将棋は自駒が多いと有利、相手駒が多いと不利であるので、自駒に正の価値、相手駒に負の価値を付け、その合計値を評価値の基準のひとつとして用いる。リーダーは最前列まで進むと勝ちであるので、前進するにつれてその価値を大きくする。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。
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、前述の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。

3.1.3 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。
本研究のプログラムでは、駒を移動させたとき、その駒が効いている各マスを調べ、相手のリーダーがいれば

王手だと判定する。

3.1.4 勝敗の判定

プログラムではリーダーが王手が掛かっている場合、王手を回避できる手以外は無効な手として、着手可能手のリストから削除される。無効な手を除去した結果、着手可能手が無くなった場合は、王手から逃れられないことを意味しており、詰みとなる。また、アンパンマン将棋ではリーダーが端に到着した場合も勝利となるが、この判定は各リーダーの y 座標の値がアンパンマンであれば1、バイキンマンであれば5になればゴールとみなし勝利が確定となるようにしている。

3.1.5 千日手の回避

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、先読みで得られる以前と同じ局面であるか検査し、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。ASAI は候補手の中から評価値がより高い手を指すため、ASAI は自分が有利な時、すなわち評価値が高い候補手があるときは千日手を避け、不利な時、すなわち全ての候補手が負の評価値を持っているときは積極的に千日手に持ち込むように行動する。

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

Anpanman はコンソール出力の実行クラスである。ユーザは、ゲーム開始時にアンパンマンチーム、バイキンマンチームがコンピュータか、ランダムか、プレイヤーかをここで選択する。移動可能かどうかを判定する createMove()を実行し、千日手の確認を行う Sennnichi()を実行しゲームを進行する。

3.2.2 クラス Board

Board は盤面の情報を管理するクラスである。コンストラクタで駒の生成を行い、showBoard()でコンソール上の盤の表示方法を定義する。showPiece()でコンソール上の駒の表示方法を定義している。player()、com()、ran()で、プレイヤー、コンピュータ、ランダム対戦の実行の動きを定義する。removePiece()で、駒を敵チームに取られたら、その駒を取り除くようにし、その後 Piece.move()で駒の移動を行い、移動後のマスに新しい駒をセットする。int value(int pleyerNum)では現在の評価値を設定する。勝ちが確定していたら評価値無限大、引き分けなら評価値0、負けが確定していたら評価値無限小になるようにしている。

Int value(int playerNum, intdepth)では先読み評価値を行う。ここで最も高い評価値の手を打つようにしている。

3.2.3 クラス Piece

Piece は駒についての情報を管理するクラスである。駒からみた盤面の初期化と盤面指定した場所が動けるかどうかの確認と候補探索を行う。

final static int□をコンストラクタで受け取り、swith 文で動ける方向を作る。SetInitialPosition()も同じようにコンストラクタから受け取って初期配置を行う。コンストラクタが2つあるのは、Piece(int type)で初期配置をセットし、もう一個の Piece (int type, int file, int rank)で、動くときの駒の配置をセットする。setOnBoard()と isOnBoard()で、盤上に駒があるかを true,false で確認する。isThere()は駒の座標を返す。moveable()は、他の駒を考慮して動けるか確認するメソッドであり、指定した x 座標と y 座標が候補の中にあるかを確認する。checkAndMove()で最終確認を行い、移動したい x 座標と y 座標をセットする。movableList()は、各駒の移動可能な x 座標と y 座標を返して、他の駒を考慮して自殺手を防ぐようにする。

3.2.4 クラス NextMove

NextMove は駒の種類と移動先の x 座標 y 座標で初期化するクラスである。type()は駒の種類を返すメソッドであり、nextFime()は移動先の x 座標を返すメソッドである。value()は移動した場合の盤面評価値を返すメソッドで setValue(int value)は評価値をセットするメソッドである。

4 実験結果と考察

本研究では、ASAI の性能を評価するために、ASAI と候補手からランダムに打つ AI（以下 RAI とする）との対戦を ASAI 先手、RAI 先手でそれぞれで1000回行った。ただし、ASAI の先読み手数は6手としている。対戦

成績は、ASAI 先手で969勝3負28引き分け、RAI 先手で19勝935負46引き分けであった。また、ASAI 対 ASAI を対戦したところ、先手が534勝225負241引き分けであった。この3つの対戦結果より、先手有利ではないかと推測される。

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の前準備としてアンパンマン将棋のアプリケーション開発と解析を行った。本研究では、アンパンマン将棋で対 CPU 戦ができる ASAI を作成した。

ASAI の性能を評価するために、ASAI と RAI との対戦を ASAI 先手、RAI 先手それぞれで1000回行った。ASAI の先読み手数が6手の場合、対戦成績は ASAI 先手で969勝3負28引き分け、RAI 先手で19勝935負46引き分けであった。また、ASAI 同士の対戦を1000回行った時、ASAI 先手で534勝225負241引き分けとなった。この結果より、アンパンマン将棋は先手有利であることが予想されるが、先手必勝という決定的な証拠を見つけることができなかった。

アンパンマン将棋は取り捨てであり、全ての駒は後退ができないため、千日手を除けば有限の手数で必ず勝負がつく。よって今後の課題として詰み局面を列挙し、完全解析をする事が挙げられる。完全解析済の動物将棋の可能な局面数は1,567,925,964通り[3]であるのに対し、アンパンマン将棋の局面数は7,138,560通りである。従って完全解析を行うことは充分可能であると予測される。

本研究ではできなかったアンパンマン将棋の完全解析をすることが今後の課題である。完全解析を行う手法としては、初期局面から到達可能な全局面を含んだソート済み配列を作成し、勝負がついている局面を全て列挙しする。そこから後退解析を行えば、完全解析が行えると考える。

後退解析とは勝負のついた局面の集合から開始し、勝負のついた局面の1手前の局面を求める操作を繰り返して、勝ち局面の集合も負け局面の集合がそれ以上増えなくなったら終了とする解析法である。

謝辞

本研究および本報告書を作成するにあたって石水隆先生にはご指導いただき、大変お世話になりました。また、同研究室のメンバーにも協力して頂き、誠にありがとうございます。

参考文献

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

[2] 池泰弘, Java 将棋のアルゴリズム, 工学社, (2005)

[3] 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/

[4] 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)

[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 Smalll 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] 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).

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

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

付録

Anpanman クラス

以下に3.2.1章で説明した Anpanman クラスのソースファイルを示す。

```
package anpanman;

import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Scanner;

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) {
		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 (board.sennichi()) break;
			if (isCom[0]) {
				score = board.com (0);
			} else {
				score = board.ran (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 (board.sennichi()) break;
                        if (isCom[1]) {
                                score = board.com (1);
                        } else {
                                score = board.ran (1);
                        }
                        fp.println (score); // 棋譜出力
                }
                fp.close();
        }
}
```

Board クラス

以下に3.2.2章で説明した Board クラスのソースファイルを示す。

```
package anpanman;

import java.util.ArrayList;
import java.util.Random;
import java.util.Scanner;

public class Board {
        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 boolean isChessStyleScore = true; // 棋譜表記をチェス式か将棋式か？

        public int[][] board // 将棋盤
        = { { BORDER, BORDER, BORDER, BORDER, BORDER},
                        { BORDER, HORRORMAN, BAIKINMAN, DOKINCHAN, BORDER},
                        { BORDER, EMPTY, EMPTY, EMPTY, BORDER},
                        { BORDER, EMPTY, EMPTY, EMPTY, BORDER},
                        { BORDER, EMPTY, EMPTY, EMPTY, BORDER},
                        { BORDER, CURRYPANMAN, ANPANMAN, SHOKUPANMAN, BORDER},
                        { BORDER, BORDER, BORDER, BORDER, BORDER} };
        int[][][] sennichi = new int[7][5][100];
        int nextFile; // 移動先のX座標
        int nextRank; // 移動先のY座標
        int time = 0;//手数
        Piece anpanman, shokupanman, currypanman, baikinman, horrorman, dokinchan; // 駒
        Boolean resign = false; // 投了したか
        Boolean checkmate = false; // 詰んだ(チェックメイト)か
        Boolean stalemate = false; // 詰んだ(スティールメイト)か
        ArrayList<NextMove>movableList; // 候補手のリスト
        int maxDepth = 5; // 先読みする手数の上限;

        /**
         * コンストラクタ
         * 盤上に駒を初期設定で生成
```

```
 */
public Board() {
        anpanman = new Piece(ANPANMAN);
        shokupanman = new Piece(SHOKUPANMAN);
        currypanman = new Piece(CURRYPANMAN);
        baikinman = new Piece(BAIKINMAN);
        horrorman = new Piece(HORRORMAN);
        dokinchan = new Piece(DOKINCHAN);
}

/**
 * コンストラクタ
 * 盤上に駒を指定した位置に配置
 * @param int□□ board 現在の盤
 */
public Board(int□□ board) {
        for (int r = 1; r <= 5; ++r)
                for (int f = 1; f <= 3; ++f)
                        this.board[r][f] = board[r][f];
        for (int r = 1; r <= 5; ++r) {
                for (int f = 1; f <= 3; ++f) {
                        switch (board[r][f]) {
                        case ANPANMAN:
                                anpanman = new Piece(ANPANMAN, f, r);
                                break;
                        case SHOKUPANMAN:
                                shokupanman = new Piece(SHOKUPANMAN, f, r);
                                break;
                        case CURRYPANMAN:
                                currypanman = new Piece(CURRYPANMAN, f, r);
                                break;
                        case BAIKINMAN:
                                baikinman = new Piece(BAIKINMAN, f, r);
                                break;
                        case HORRORMAN:
                                horrorman = new Piece(HORRORMAN, f, r);
                                break;
                        case DOKINCHAN:
                                dokinchan = new Piece(DOKINCHAN, f, r);
                                break;
                        }
                }
        }
}

/**
 * 盤を表示
 */
public void showBoard() {
        System.out.println("　 123");
        for (int r = 0; r < board.length; ++r) {
                switch (r) {
                case 0:
                        System.out.print("　");
                        break;
                case 1:
                        System.out.print("一");
                        break;
                case 2:
                        System.out.print("二");
                        break;
                case 3:
```

```
                        System.out.print("三");
                        break;
                case 4:
                        System.out.print("四");
                        break;
                case 5:
                        System.out.print("五");
                        break;
                case 6:
                        System.out.print("　");
                        break;
                }
                for (int f = 0; f < board[r].length; ++f) {
                        System.out.print(showPiece(board[r][f]));
                }
                System.out.println();
        }
}

/**
 * 駒を表示
 * @param 駒の種類
 * @return 駒の文字列表現
 */
public String showPiece(int type) {
        switch (type) {
        case ANPANMAN:
                return "ア";
        case SHOKUPANMAN:
                return "食";
        case CURRYPANMAN:
                return "カ";
        case BAIKINMAN:
                return "バ";
        case HORRORMAN:
                return "ホ";
        case DOKINCHAN:
                return "ド";
        case EMPTY:
                return "　";
        case BORDER:
                return "■";
        default:
                return "？";
        }
}

/**
 * 各プレイヤーの手番
 * @paran int playerNum プレイヤー番号
 * @return 指した手の棋譜
 */
public String player(int playerNum) {
        Scanner keyBoardScanner = new Scanner(System.in);
        String inputPiece; // 駒の種類入力用
        Piece piece; // 動かす駒
        int type; // 動かす駒の種類
        int nextFile, nextRank; // 移動先
        ArrayList<NextMove> onesMovableList; // ある駒が移動可能な手のリスト

        while (true) { // 適切な駒が選択されるまでループ
                if (playerNum == 0) {
```

```
            System.out.println("アンパンマンチームの番です");
            System.out.print("進める駒(A,S,C)を選んでください(R=投了):");
} else {
            System.out.println("バイキンマンチームの番です");
            System.out.print("進める駒(B,H,D)を選んでください(R=投了):");
}
inputPiece = keyBoardScanner.next();
if (inputPiece.equals("A") || inputPiece.equals("a")) {
            piece = anpanman;
            type = ANPANMAN;
} else if (inputPiece.equals("S") || inputPiece.equals("s")) {
            piece = shokupanman;
            type = SHOKUPANMAN;
} else if (inputPiece.equals("C") || inputPiece.equals("c")) {
            piece = currypanman;
            type = CURRYPANMAN;
} else if (inputPiece.equals("B") || inputPiece.equals("b")) {
            piece = baikinman;
            type = BAIKINMAN;
} else if (inputPiece.equals("H") || inputPiece.equals("h")) {
            piece = horrorman;
            type = HORRORMAN;
} else if (inputPiece.equals("D") || inputPiece.equals("d")) {
            piece = dokinchan;
            type = DOKINCHAN;
} else if (inputPiece.equals("R") || inputPiece.equals("r")) {
            System.out.println("投了します");
            resign = true;
            if (isChessStyleScore) {
                        if (playerNum == 0) {
                                    return "1-0";
                        } else {
                                    return "0-1";
                        }
            } else {
                        return "投了";
            }
} else {
            if (playerNum == 0) {
                        System.out.println("A,S,C から選んでください");
            } else {
                        System.out.println("B,H,D から選んでください");
            }
            continue; // 選び直し
}

if (playerNum == 0
&& !(type == ANPANMAN || type == SHOKUPANMAN || type == CURRYPANMAN)) {
            System.out.println("A,S,C から選んでください");
            continue; // 選び直し
} else if (playerNum == 1
&& !(type == BAIKINMAN || type == HORRORMAN || type == DOKINCHAN)) {
            System.out.println("B,H,D から選んでください");
            continue; // 選び直し
}

if (piece == null) {
            System.out.println(name(type) + "はすでに取られています");
            continue; // 選び直し
}

onesMovableList = new ArrayList<NextMove>(); // 指定した駒が移動可能な手のリスト
```

```
                for (int i = 0; i < movableList.size(); ++i) {
                        if (movableList.get(i).type() == type) { // 指定した駒を動かす手か？
                                onesMovableList.add(movableList.get(i));
                                                // 指定した駒が移動可能な手の数を加える
                        }
                }
                if (onesMovableList.size() == 0) { // 指定した駒が移動可能な位置が無い場合
                        System.out.println(name(type) + "が進める位置はありません");
                        continue; // 選び直し
                }
                System.out.println(name(type) + "が進む場所を選んでください");
                System.out.print(name(type) + "は現在(" + piece.file() + "," + piece.rank() + ")にいて");
                for (int i = 0; i < onesMovableList.size(); ++i) {
                        System.out.print("(" + onesMovableList.get(i).nextFile() + ","
                        +  onesMovableList.get(i).nextRank()
                                        + ")");
                }
                System.out.println("へ移動できます");

                System.out.print("x座標 (1～3):");
                nextFile = keyBoardScanner.nextInt();
                System.out.print("y座標 (1～5):");
                nextRank = keyBoardScanner.nextInt();

                boolean movable = false; // 指定した位置に移動可能か
                for (int i = 0; i < onesMovableList.size(); ++i) {
                        if ((nextFile == onesMovableList.get(i).nextFile())
                         && (nextRank == onesMovableList.get(i).nextRank())) {
                                movable = true;
                                break;
                        }
                }
                if (!movable) {
                        System.out.println(name(type) + "は(" + nextFile + "," +  nextRank
                        + ")へは移動できません");
                        continue; // 選び直し
                }
                break; // while ループから脱出
        }
        time++;
        return movePiece(piece, type, nextFile, nextRank); // 駒を移動させる
}

public String ran(int playerNum) {
        int type, nextFile, nextRank; // 移動する駒の種類, 移動先の座標
        Piece piece = null; // 移動する駒
        Random rnd = new Random();
        int ran = rnd.nextInt(movableList.size());
        NextMove now = movableList.get(ran);
        type = now.type(); // 移動する駒の種類
        nextFile = now.nextFile(); // 移動先のX座標
        nextRank = now.nextRank(); // 移動先のY座標
        if (movableList.size() == 0) {
                System.out.print("投了します。");
        }
        switch (type) {
        case ANPANMAN:
                piece = anpanman;
                break;
        case SHOKUPANMAN:
                piece = shokupanman;
                break;
```

```
            case CURRYPANMAN:
                piece = currypanman;
                break;
            case BAIKINMAN:
                piece = baikinman;
                break;
            case HORRORMAN:
                piece = horrorman;
                break;
            case DOKINCHAN:
                piece = dokinchan;
                break;
            }
            time++;
            return movePiece(piece, type, nextFile, nextRank); // 駒を移動させる
}

/**
 * COMの手番
 * @paran int playerNum プレイヤー番号
 * @return 指した手の棋譜
 */
public String com(int playerNum) {
            int type, nextFile, nextRank; // 移動する駒の種類, 移動先の座標
            Piece piece = null; // 移動する駒

            int bestValue; // 最も良い盤面の評価値
            NextMove bestMove = null; // 最も良い手
            int nextPlayerNum; // 次の手番プレイヤー

            if (playerNum == 0) { // アンマンパンチームの場合 評価値が高いほど良い手と見做す
                        nextPlayerNum = 1; // 次の手番プレイヤー
                        bestValue = Integer.MIN_VALUE;
                        for (int i = 0; i < movableList.size(); ++i) {
                                    NextMove nextMove = movableList.get(i); // i番目の候補手
                                    Board nextBoard = nextBoard(nextMove, nextPlayerNum); // 次の盤面を生成
                                    int value = nextBoard.value(nextPlayerNum, maxDepth);// 盤面の評価値を計算
                                    if (value > bestValue) { // 高評価の手を発見した
                                                bestMove = nextMove; // 高評価の手を記憶
                                                bestValue = value; // 評価値を記憶
                                    }
                                    if (bestValue == Integer.MAX_VALUE) { // 評価無限大の手＝必勝の手を発見
                                                break; // ループ脱出
                                    }
                        }
                        if (bestValue == Integer.MIN_VALUE) { // 評価値無限小の手しか無い＝負け確定
                                    System.out.println("投了します");
                                    resign = true;
                                    if (isChessStyleScore) {
                                                return "1-0";
                                    } else {
                                                return "投了";
                                    }
                        }
            } else { // バイキンマンチームの場合 評価値が低いほど良い手と見做す
                        nextPlayerNum = 0; // 次の手番プレイヤー
                        bestValue = Integer.MAX_VALUE;
                        for (int i = 0; i < movableList.size(); ++i) {
                                    NextMove nextMove = movableList.get(i); // i番目の候補手
                                    Board nextBoard = nextBoard(nextMove, nextPlayerNum); // 次の盤面を生成
                                    int value = nextBoard.value(nextPlayerNum, maxDepth);// 盤面の評価値を計算
                                    if (value < bestValue) { // 高評価の手を発見した
```

```
                                bestMove = nextMove; // 高評価の手を記憶
                                bestValue = value; // 評価値を記憶
                        }
                        if (bestValue == Integer.MIN_VALUE) { // 評価無限小の手＝必勝の手を発見
                                break; // ループ脱出
                        }
                }
                if (bestValue == Integer.MAX_VALUE) { // 評価値無限大の手しか無い＝負け確定
                        System.out.println("投了します");
                        resign = true;
                        if (isChessStyleScore) {
                                return "0-1";
                        } else {
                                return "投了";
                        }
                }
        }
        type = bestMove.type(); // 移動する駒の種類
        nextFile = bestMove.nextFile(); // 移動先のX座標
        nextRank = bestMove.nextRank(); // 移動先のY座標

        switch (type) {
        case ANPANMAN:
                piece = anpanman;
                break;
        case SHOKUPANMAN:
                piece = shokupanman;
                break;
        case CURRYPANMAN:
                piece = currypanman;
                break;
        case BAIKINMAN:
                piece = baikinman;
                break;
        case HORRORMAN:
                piece = horrorman;
                break;
        case DOKINCHAN:
                piece = dokinchan;
                break;
        }
        time++;
        return movePiece(piece, type, nextFile, nextRank); // 駒を移動させる
}

/**
 * 指定した位置に駒を移動させ、その棋譜表記を返す
 * @param Piece piece 移動させる駒
 * @param int type 移動させる駒の種類
 * @param int nextFile 移動先のX座標
 * @param int nextRank 移動先のY座標
 */
public String movePiece(Piece piece, int type, int nextFile, int nextRank) {
        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;
                }
                if (board[nextRank][nextFile] != EMPTY)
                        score += "x";
                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;
                }
            }
            if (board[nextRank][nextFile] != EMPTY) { // 移動先に駒がある場合
                removePiece(nextFile, nextRank); // 移動先にある駒を取り除く
            }

            board[piece.rank()][piece.file()] = EMPTY; // 移動前のマスを空白に
            piece.move(nextFile, nextRank); // 駒を移動
            board[piece.rank()][piece.file()] = type; // 移動後のマスを指定した駒に

            return score;
    }

    /**
     * 指定した位置にある駒を盤から取り除く
```

```
 * @param int file X座標
 * @param int rank Y座標
 */
public void removePiece(int nextFile, int nextRank) {
	switch (board[nextRank][nextFile]) {
	case ANPANMAN: // 移動先にアンパンマン
		anpanman = null; // アンパンマンを取り除く
		break;
	case SHOKUPANMAN: // 移動先に食パンマン
		shokupanman = null; // 食パンマンを取り除く
		break;
	case CURRYPANMAN: // 移動先にカレーパンマン
		currypanman = null; // カレーパンマンを取り除く
		break;
	case BAIKINMAN: // 移動先にバイキンマン
		baikinman = null; // バイキンマンを取り除く
		break;
	case HORRORMAN: // 移動先にホラーマン
		horrorman = null; // ホラーマンを取り除く
		break;
	case DOKINCHAN: // 移動先にドキンちゃん
		dokinchan = null; // ドキンちゃんを取り除く
		break;
	}
}

/**
 * 千日手をチェックする
 */
public boolean sennichi() {
	int count=0;
	int check=0;
	for (int x = 0; x < 5; x++) {
		for (int y = 0; y < 7; y++) {
			sennichi[y][x][time] = board[y][x];
		}
	}

	for (int i = 0; i < time; i++) {
		count=0;
		for (int x = 0; x < 5; x++) {
			for (int y = 0; y < 7; y++) {
				if (sennichi[y][x][i] == board[y][x]) {
					count++;
				}
			}
		}
		if(count==35){
			check++;
		}
	}
	System.out.println(check);
	if(check>=3){
		System.out.println("千日手のため引き分け");
		return true;
	}
	return false;
}

/**
 * 勝負がついたか
 * @param int playerNum プレイヤー番号
```

```
 * @return 勝負がついたか
 */
public boolean checkWin(int playerNum) {
	if (playerNum == 0) { // アンパンマンチームの手番
		if (baikinman == null) { // バイキンマンを取った
			System.out.println("アンパンマンチームの勝利！");
			return true;
		} else if (anpanman.rank() == 1) { // アンパンマンがゴールした
			System.out.println("アンパンマンチームの勝利！");
			return true;
		} else if (isMate(1)) { // バイキンマンが詰んだ
			if (isChecked(1)) { // バイキンマンに王手がかかっている
				System.out.println("チェックメイト！");
				System.out.println("アンパンマンチームの勝利！");
			} else {
				System.out.println("スティールメイト！");
				System.out.println("引き分けです");
			}
			return true;
		} else if (resign) { // 投了した
			System.out.println("バイキンマンチームの勝利！");
			return true;
		} else {
			return false;
		}
	} else { // バイキンマンチームの手番
		if (anpanman == null) { // アンパンマンを取った
			System.out.println("バイキンマンチームの勝利！");
			return true;
		} else if (baikinman.rank() == 5) { // バイキンマンがゴールした
			System.out.println("バイキンマンチームの勝利！");
			return true;
		} else if (isMate(0)) { // アンパンマンチームが詰んだ
			if (isChecked(0)) { // アンパンマンに王手がかかっている
				System.out.println("チェックメイト！");
				System.out.println("バイキンマンチームの勝利！");
			} else {
				System.out.println("スティールメイト！");
				System.out.println("引き分けです");
			}
			return true;
		} else if (resign) { // 投了した
			System.out.println("アンパンマンチームの勝利！");
			return true;
		} else {
			return false;
		}
	}
}

/**
 * 現在王手がかかっているか
 * @param int playerNum プレイヤー番号
 * @return 王手がかかっているか
 */
public boolean isChecked(int playerNum) {
	int file, rank;
	if (playerNum == 0) {
		file = anpanman.file(); // アンパンマンの座標
		rank = anpanman.rank();
		if (board[rank][file - 1] == HORRORMAN)
			return true; // ホラーマンが王手
```

```
                else if (board[rank - 1][file] == HORRORMAN)
                        return true; // ホラーマンが王手
                else if (board[rank][file + 1] == HORRORMAN)
                        return true; // ホラーマンが王手
                else if (board[rank - 1][file - 1] == DOKINCHAN)
                        return true; // ドキンちゃんが王手
                else if (board[rank - 1][file] == DOKINCHAN)
                        return true; // ドキンちゃんが王手
                else if (board[rank - 1][file + 1] == DOKINCHAN)
                        return true; // ドキンちゃんが王手
                else
                        return false;
        } else {
                file = baikinman.file(); // バイキンマンの座標
                rank = baikinman.rank();
                if (board[rank][file - 1] == SHOKUPANMAN)
                        return true; // 食パンマンが王手
                else if (board[rank + 1][file] == SHOKUPANMAN)
                        return true; // 食パンマンが王手
                else if (board[rank][file + 1] == SHOKUPANMAN)
                        return true; // 食パンマンが王手
                else if (board[rank + 1][file - 1] == CURRYPANMAN)
                        return true; // カレーパンマンが王手
                else if (board[rank + 1][file] == CURRYPANMAN)
                        return true; // カレーパンマンが王手
                else if (board[rank + 1][file + 1] == CURRYPANMAN)
                        return true; // カレーパンマンが王手
                else
                        return false;
        }
}

/**
 * 詰みの判定
 * 移動可能な手が無ければ詰み(チェックメイトまたはスティールメイト)
 * @param int playerNum プレイヤー番号
 * @return 詰んだか
 */
public boolean isMate(int playerNum) {
        return (movableList.size() == 0); // 可能な手が無ければ詰み
}

/**
 * 候補手の作成
 * また, 移動可能な手のリストを movableList に保持する
 * @param int playerNum プレイヤー番号
 */
public void createMovableList(int playerNum) {
        if (playerNum == 0) { // アンパンマンチームの場合
                movableList = anpanman.movableList(board); // アンパンマンが移動可能な手
                if (shokupanman != null) { // 盤上にまだ食パンマンがある場合
                        movableList.addAll(shokupanman.movableList(board)); // 食パンマンが移動可能な手
                }
                if (currypanman != null) { // 盤上にまだカレーパンマンがある場合
                        movableList.addAll(currypanman.movableList(board)); // カレーパンマンが移動可能な手
                }
                if (isChecked(0)) { // アンパンマンに王手が掛かっている場合
                        int nextPlayerNum = (playerNum == 0) ? 1 : 0; // 次のプレイヤー番号
                        for (int i = 0; i < movableList.size();) {
                                Board nextBoard = nextBoard(movableList.get(i), playerNum);
                                if (nextBoard.isChecked(0)) { // 移動後も王手が掛かっている手は無効
                                        movableList.remove(i); // 移動後も王手が掛かっている手を取り除く
```

```
                                        } else {
                                                ++i;
                                        }
                                }
                        }
                } else { // バイキンマンチームの場合
                        movableList = baikinman.movableList(board); // バイキンマンが移動可能な手
                        if (horrorman != null) { // 盤上にまだホラーマンがある場合
                                movableList.addAll(horrorman.movableList(board)); // ホラーマンが移動可能な手
                        }
                        if (dokinchan != null) { // 盤上にまだドキンちゃんがある場合
                                movableList.addAll(dokinchan.movableList(board)); // ドキンちゃんが移動可能な手
                        }
                        if (isChecked(1)) { // バイキンマンに王手が掛かっている場合
                                for (int i = 0; i < movableList.size();) {
                                        Board nextBoard = nextBoard(movableList.get(i), playerNum);
                                        if (nextBoard.isChecked(1)) { // 移動後も王手が掛かっている手は無効
                                                movableList.remove(i); // 移動後も王手が掛かっている手を取り除く
                                        } else {
                                                ++i;
                                        }
                                }
                        }
                }
        }

        /**
         * 現在の盤の評価値を表示
         * 先読みは行わず現在の盤面のみで評価する
         * @param int playerNum プレイヤー番号
         * @return 評価値
         */
        public int value(int playerNum) {
                checkmate = false;
                stalemate = false;

                /* 現時点ですでに詰んでいるかどうかのチェック */
                if (playerNum == 0) { // アンパンマンチームの手番
                        if (anpanman == null) { // アンパンマンが取られた
                                checkmate = true;
                                return Integer.MIN_VALUE; // 評価値無限小
                        } else if (baikinman.rank() == 5) { // バイキンマンがゴールした
                                checkmate = true;
                                return Integer.MIN_VALUE; // 評価値無限小
                        } else if (isMate(0)) { // アンパンマンチームが詰んだ
                                if (isChecked(0)) { // アンパンマンに王手がかかっている
                                        checkmate = true;
                                        return Integer.MIN_VALUE; // 評価値無限小
                                } else { // ステールメイト
                                        stalemate = true;
                                        return 0; // 評価値0
                                }
                        } else if (resign) { // バイキンマンチームが投了した
                                return Integer.MAX_VALUE; // 評価値無限大
                        }
                } else { // バイキンマンチームの手番
                        if (baikinman == null) { // バイキンマンが取られた
                                checkmate = true;
                                return Integer.MAX_VALUE; // 評価値無限大
                        } else if (anpanman.rank() == 1) { // アンパンマンがゴールした
                                checkmate = true;
                                return Integer.MAX_VALUE; // 評価値無限大
```

```
        } else if (isMate(1)) { // バイキンマンが詰んだ
                if (isChecked(1)) { // バイキンマンに王手がかかっている
                        checkmate = true;
                        return Integer.MAX_VALUE; // 評価値無限大
                } else { // ステールメイト
                        stalemate = true;
                        return 0; // 評価値0
                }
        } else if (resign) { // アンパンマンチームが投了した
                return Integer.MIN_VALUE; // 評価値無限小
        }
}

/* 現時点ではまだ詰んでいない場合 */
int value = 0;
switch (anpanman.rank()) { // アンパンマンはゴールに近い方が高評価
case 1: // アンパンマンがゴールした場合
        return Integer.MAX_VALUE; // すでにチェックしているのでここに処理が移ることはありえない
case 2:
        value += 13;
        break;
case 3:
        value += 9;
        break;
case 4:
        value += 7;
        break;
case 5:
        value += 6;
        break;
}
if (shokupanman != null) { // 盤上に食パンマンがある場合
        if (shokupanman.rank() == 1) // 食パンマンは端まで進むと価値が下がる
                value += 1;
        else
                value += 2;
}
if (currypanman != null) { // 盤上にカレーパンマンがある場合
        if (currypanman.rank() == 1) // カレーパンマンは端まで進むと価値が下がる
                value += 0;
        else
                value += 4;
}
switch (baikinman.rank()) { // バイキンマンはゴールに近い方が高評価
case 5: // バイキンマンがゴールした場合
        return Integer.MIN_VALUE; // すでにチェックしているのでここに処理が移ることはありえない
case 4:
        value -= 13;
        break;
case 3:
        value -= 9;
        break;
case 2:
        value -= 7;
        break;
case 1:
        value -= 6;
        break;
}
if (horrorman != null) { // 盤上にホラーマンがある場合
        if (horrorman.rank() == 5) // ホラーマンは端まで進むと価値が下がる
                value -= 1;
```

```
                else
                        value -= 2;
        }
        if (dokinchan != null) { // 盤上にドキンちゃんがある場合
                if (dokinchan.rank() == 5) // ドキンちゃんは端まで進むと価値が下がる
                        value -= 0;
                else
                        value -= 4;
        }
        if (playerNum == 0) {
                value += movableList.size(); // 移動可能な手が多いほど高評価値
        } else {
                value -= movableList.size(); // 移動可能な手が多いほど低評価値
        }
        return value;
}

/**
 * 現在の盤の評価値を表示
 * @param int playerNum プレイヤー番号
 * @param int depth 先読みする手数
 * @return 評価値
 */
public int value(int playerNum, int depth) {
        createMovableList(playerNum); // 移動可能な手のリストを生成

        int value = value(playerNum); // 先読み無しの現在の盤面の評価値を求める
        if (depth == 0) {
                return value;
        }
        if (checkmate || stalemate || resign)
                return value; // すでに詰んでいるときはそのまま値を返す

        int bestValue; // 最も良い評価値
        NextMove bestMove = null; // 最も良い手
        int nextPlayerNum; // 次の手番プレイヤー

        if (playerNum == 0) { // アンパンマンチームの場合 評価値が高いほど良い手と見做す
                nextPlayerNum = 1; // 次の手番プレイヤー
                bestValue = Integer.MIN_VALUE; // 盤面の評価値の初期値

                for (int i = 0; i < movableList.size(); ++i) {
                        NextMove nextMove = movableList.get(i); // i番目の候補手
                        Board nextBoard = nextBoard(nextMove, nextPlayerNum); // 次の盤面を生成
                        value = nextBoard.value(nextPlayerNum, depth - 1);// 盤面の評価値を計算
                        if (value > bestValue) { // 高評価の手を発見した
                                bestMove = nextMove; // 高評価の手を記憶
                                bestValue = value; // 評価値を記憶
                        }
                        if (bestValue == Integer.MAX_VALUE) { // 評価無限大の手＝必勝の手を発見
                                return Integer.MAX_VALUE;
                        }
                }
        } else { // バイキンマンチームの場合 評価値が低いほど良い手と見做す
                nextPlayerNum = 0; // 次のプレイヤー番号
                bestValue = Integer.MAX_VALUE; // 盤面の評価値の初期値

                for (int i = 0; i < movableList.size(); ++i) {
                        NextMove nextMove = movableList.get(i); // i番目の候補手
                        Board nextBoard = nextBoard(nextMove, nextPlayerNum); // 次の盤面を生成
                        value = nextBoard.value(nextPlayerNum, depth - 1);// 盤面の評価値を計算
                        if (value < bestValue) { // 高評価の手を発見した
```

```
                                        bestMove = nextMove; // 高評価の手を記憶
                                        bestValue = value; // 評価値を記憶
                                }
                                if (bestValue == Integer.MIN_VALUE) { // 評価無限小の手＝必勝の手を発見
                                        return Integer.MIN_VALUE;
                                }
                        }
                }
                /*Random rnd = new Random();
                int ran = rnd.nextInt (3); // 0～2の乱数を生成
                value = bestValue + ran - 1; // 評価値に乱数を加える
                return value;*/
                value = bestValue;
                return value;
        }

        /**
         * 指定した駒を指定した位置に動かした後の盤を得る
         * @param NextMove nextMove 次の手
         * @param int playerNum プレイヤー番号
         * @return 移動した後の盤
         */
        public Board nextBoard(NextMove nextMove, int playerNum) {
                Board nextBoard = new Board(board);
                int movingType = nextMove.type(); // 移動する駒の種類
                Piece movingPiece = null; // 移動する駒
                switch (movingType) {
                case ANPANMAN:
                        movingPiece = nextBoard.anpanman;
                        break;
                case SHOKUPANMAN:
                        movingPiece = nextBoard.shokupanman;
                        break;
                case CURRYPANMAN:
                        movingPiece = nextBoard.currypanman;
                        break;
                case BAIKINMAN:
                        movingPiece = nextBoard.baikinman;
                        break;
                case HORRORMAN:
                        movingPiece = nextBoard.horrorman;
                        break;
                case DOKINCHAN:
                        movingPiece = nextBoard.dokinchan;
                        break;
                }
                int currentFile = movingPiece.file(); // 移動する駒の現在のX座標
                int currentRank = movingPiece.rank(); // 移動する駒の現在のX座標
                int nextFile = nextMove.nextFile(); // 移動先のX座標
                int nextRank = nextMove.nextRank(); // 移動先のY座標
                nextBoard.movePiece(movingPiece, movingType, nextFile, nextRank); // 駒を移動させる

                return nextBoard;
        }

        /**
         * 駒名を返す
         * @param int type 駒の種類
         * @return 駒名
         */
        public String name(int type) {
                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 "？";
                }
        }
}
```

Piece クラス

以下に3.2.3章で説明した Piece クラスのソースファイルを示す。

```
package anpanman;

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;
	}
}
```

NextMove クラス

以下に3.2.4章で説明した NextMove クラスのソースファイルを示す。

```
package anpanman;

/**
 * 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


	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;
	}
}
```