卒業研究報告書

題目

拡張どうぶつ将棋の解析

指 導 教 員

石 水　隆　講師

報告者

10-1-037-0021

赤井　隆純

近畿大学理工学部情報学科

平成 24 年 1 月 31 日提出

概要

「ごろごろどうぶつしょうぎ」[1]（以下ごろごろ動物将棋とする）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。ごろごろ動物将棋は、「５ｘ６の盤面」と、それぞれ本将棋の玉将、金将、銀将、歩兵に相当するライオン、犬、猫、ひよこの４種類の駒を使用する。ごろごろ動物将棋は、ライオンを取ると勝ちとなり、その他のルールは本将棋と同じである。

本研究では、ごろごろ動物将棋の完全解析を最終目標とし、ごろごろ動物将棋プログラムを ~~j~~Java を用いて作成する。ただし、それに先立ち持ち駒なしの成り駒なしのでの簡易版を作成した。

本研究の AI は、ある局面における全ての合法手に対して、数手先までの先読みを行い、その結果を元に各手に評価値を設定し、価値が最大である手を指す。評価値の決定は、盤上の駒の数と着手可能手数により行う。評価値の計算は、着手可能手に対して再帰的に計算を行い、先読み手数が一定値に達した場合は、その局面での評価値を求め、着手可能手を指した次の局面の評価値のうち最大であるものをその局面での評価値とする。AI は、着手可能手のうち最も高い評価値が得られる手を指す。

本研究で作成した AI 同士の対戦を 100 回行ったところ、先手の 43 戦 36 敗 21 引き分けであった。持ち駒と成り駒なしなので千日手が多数発生し、引き分けが多い結果となった。ただ、現状況においては AI 同士の対戦の将譜と詰み局面から先手が有利ではないかと推測される。

目次

1. 序論

1. 本研究の背景

ごろごろ動物将棋は2012年11月15日に発売された女流棋士北尾まどか初段によって考案された二人零和有限確定完全情報ゲームである。

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、勝ちを＋１、負けをー１、引き分けを０として、全プレイヤーの数値が合計常に０（一定）となるゲームのことである。有限とは、各プレイヤーの可能な手の組み合わせ総数が有限であるゲームのことである。確定とは、プレイ中のゲームに偶然の要素が起こらないことである。ただし、先手後手を決めるために行う振り駒などは含まれない。完全情報ゲームとは、両プレイヤーが局面の情報および、各プレイヤーがお互いのこれまでに行った選択について全ての情報を知ることができるゲームのことである。二人零和有限確定完全情報ゲームは、その性質上解析を行い易いため、現在では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開発されている[2][3][4]。

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指すことができる。しかし多くのボードゲームでは,総局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、総局数はリバーシが 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、総局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（五目並べの一種）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]。囲碁は、サイズ 4x4 では双方最善手を打つと持碁(引き分け)になり[8]、5x5 の囲碁は黒の 25 目勝ちとなる[9]。

将棋については、盤面のサイズや駒数を減らした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 5x6 の盤と４種類の駒を使用する。図１、２にそれぞれの初期盤面を示す。総局数はそれぞれ５五将棋がおよそ $10x10^{17}$ 通り、ゴロゴロ将棋が $10x10^{21}$ 通りあり、本将棋と比べて総局数が少ないが現在のところまだ完全解析されていない。

図１５五将棋の初期盤面

図２ ゴロゴロ将棋の初期盤面

完全解析されているミニ将棋として、どうぶつしょうぎがある（以下動物将棋とする）[12]。動物将棋はサイズ 3x4 の盤とそれぞれ本将棋の玉将、角行、飛車、歩兵に相当するライオン、象、キリン、ひよこの４種類の駒（象、キリンは１マスのみ移動）を使用する。図３に動物将棋の初期盤面を示す。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[13]。

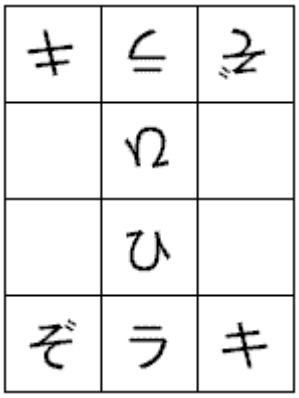

図３ 動物将棋の初期盤面

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、終盤での詰み読み、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事が出来る。

局面の評価値計算とは、現在の局面が有利か不利かというのを数値化して皮革するための計算である。評価値の設定は重要であり、もし誤った評価値を設定していた場合、駒がただでとられてしまうなどの駒損となる不利な手を指してしまう。

定跡データベースとは、序盤での決まった指し手の形のことである。序盤戦は優劣がつきにくい場合が多いので定跡通りに進めることができるのは非常に有利である。この定跡をデータベースに保存しておくことで、その局面がきた場合には定 跡通りに指そうとするので、時間の短縮にも繋がる。

対戦データベースとは、繰り返し対戦を行う場合に、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である。過去の対戦において、その手が有効であったかを対戦記録から判定し、データ

ベースに蓄える。数多く対戦を行うことによって、その手の精度が高いと判定することができる。

一定手数の先読みとは、ゲーム木を一定数の深さまで探索しそこで局面の評価を行う。先読みの基本原理は対戦相手が最善の手を差し返してくるということ前提の min-max 戦略である。

終盤での詰み読みとは、ゲームが終盤に差し掛かると詰みまでの局面数が限られてくるので、勝敗がつくまでの読み切りが可能となってくる。

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。ボードゲームへの応用としては、ある局面からランダムに手を指し続けたときの勝率を計算して、勝率が高い局面ほど優れた局面と判断する。さらにゲームの性質上有利とされる手が存在するならばその手を採用するなどの知識を混合させて行うのが一般的であるといえる。

以上の手法を用いることで、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事が可能であり、ゲームによってはプロに勝つことも可能である。

チェスでは、1997 年 5 月にチェスプログラム「Deep Blue」[14]が世界チャンピオン Garry Kimovich Kasparov 氏と対戦を行い 2 勝 1 敗 3 引き分けで勝利した[2]。将棋では、将棋プログラム「ボンクラーズ」が[15]が 2012 年 1 月 14 日に開催されたプロ棋士とコンピュータ将棋ソフトによる将棋戦、第 1 回将棋電王戦[17]で元プロ棋士の米長邦永世棋聖と対戦しボンクラーズが先手 113 手で勝利した[3]。また、2013 年 3 月に第 2 回将棋電王戦五番勝負（団体戦）が開催され、コンピュータ将棋側の 3 勝 1 敗 1 引き分けとなった[17]。しかし、同年 12 月 31 日に第 2 回電王戦で将棋プログラム「ツツカナ」[18]に敗北した船江恒平五段が、リベンジマッチを挑み、船江五段が先手 85 手で勝利した[19]。オセロでは、オセロプログラム「ロジステロ」[16]が元日本チャンピオン富永健太氏の 2 番勝負が行われ、ロジステロ先手 38 対 26 でロジステロの 12 目勝ち、冨永氏先手 23 対 41 でロジステロの 18 目勝ちとなり、ロジステロが 2 勝した[4]。

4. 本研究の目的

前述の通り、動物将棋は完全解析されており、双方最善手を指すと後手勝ちとなることが判明している。その拡張版となるごろごろどうぶつしょうぎ[1]（以下ごろごろ動物将棋とする）は 2012 年 11 月 15 日に発売されたが、未だ完全解析がされていない。そこで本研究ではごろごろ動物将棋の完全解析を目指す。

5. 本報告書の構成

本報告書の構成は以下の通りである。第 2 章において、本研究が対象とするごろごろ動物将棋について説明する。続く第 3 章で、ごろごろ動物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。第 4 章では実験結果と考察を述べ、第 5 章では結論と今後の課題を述べる。

2. ごろごろ動物将棋

本節ではごろごろ動物将棋について説明する。

1. ごろごろ動物将棋の将棋盤と駒

ごろごろ動物将棋は、サイズ5x6の将棋盤と、それぞれ本将棋の玉将、金将、銀将、歩兵に相当するライオン、犬、猫、ひよこの4種類の駒を使用する。以下では、簡単のためにごろごろ動物将棋の駒はすべて対応する本将棋の駒で表す。図2にごろごろ動物将棋の初期盤面を示す。

2. ごろごろ動物将棋の進行と勝敗

ごろごろ動物将棋は、本将棋と同様に 2 人のプレイヤーで遊ぶゲームであり、成り駒や持ち駒も存在している。盤面手前が先手で奥が後手である。成り駒については、先手の銀将、歩兵は盤面の 2 段目まで進めばそれぞれ金将へと成り、後手の銀将と歩兵は盤面の 5 段目まで進めばそれぞれ金将へと成る。動物将棋のようなトライルールはない。反則手として、歩兵を縦の列に複数指す二歩や、そこから移動できない最前列（先手の場合は 1 段目、後手の場合は 6 段目）に歩を指すこと、最前列に歩を進めたときに成らないことなどがある。また、本将棋と同様に、盤上の歩兵を進めて相手玉を詰ませる突き歩詰め可能であるが、持ち駒から指した歩で詰ませるのは打ち歩詰めとなり反則手となる。同一局面が 3 回起こった場合は千日手とみなし、先手後手を入れ替え指し直しになる。しかし、連続王手で千日手が発生した場合には、王手をかけた側が指す手を変えない限り、かけた側の負けとなる。ステイルメイトに関しては、将棋で起こる場合は一様に大差がついた局面で、優勢側がわざと詰みに行かない場合などに限られるために皆無となる。しかし、理論上では存在するので、ステイルメイトが発生した場合には、受け手側の負けとする。また、双方入玉時の場合には、本将棋では、大駒（飛車や角行）は 5 点、玉将を 0 点、小駒（大駒、玉将以外の駒）を 1 点とし盤上および持ち駒の合計点を求め、双方 24 点以上であれば引き分けとし、23 点以下であれば点数の低い方を負けとする。ただし、アマチュアの大会等で、引き分けはでは不具合が生じる場合には、27 点以上ある方を勝ちとし、双方 27 点で同点の場合は、後手勝ちとする「27 点法」が用いられる場合もある。ごろごろ動物将棋の場合は、初期状態で金将が 2 点、銀将が 2 点、歩兵が 3 点（1 つで 1 点）の合計 7 点であり、仮に本将棋の「27 点法」に相当する「7 点法」を採用した場合は、上記の計算で、7 点以上あるほうが勝ち、同点の場合には後手勝ちとなる。

3. ごろごろ動物将棋の総局数

ごろごろ動物将棋の総局数について考える。まず先手の玉将は必ず盤上に存在するので 30 通り、後手の玉将は先手の玉将以外のマスのどこかに存在するので 29 通り、よって双方合わせて 30×29 通りとなる。次に 4 つある金将は、それぞれ先手の駒として盤上にあるのが 28 通り、後手の駒として盤上にあるのが 28 通り、そして先手の持ち駒と後手の持ち駒が各 1 通り、合わせて 58 通り。これが 4 つあるので 58^4 通り、ただし 4 つの駒に区別はないので、$58^4/24$ 通りとなる。銀将も同様に求めると $58^4/24$ 通りとなり、6 つある歩兵も同様に求めることができるので、$58^6/720$ 通りとなる。以上のことから、ごろごろ動物将棋の総局数は、およそ 1.0×10^{22} 通りである。

3. 研究内容

本研究では、完全解析に先立ち Java を用いてごろごろ動物将棋の対人、対 AI 戦が可能なプログラムを作成する。ただし、それに先立ち持ち駒なし成り駒なしでの簡易版を作成した。対 AI については、各着手可能手から得られる局面を先読みし、それを元に各手の評価値を求め、最も評価の高い手を指すようにしている。また、ごろごろ動物将棋の反則手については、2.2 節で述べたうち、二歩や最前列へ移動した際の歩兵の不成り、打ち歩詰め、双方入玉時の処理が未対応である。

1. ごろごろ動物将棋 AI 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．本節ではごろごろ動物将棋 AI で用いた手法について説明する。

1. 着手可能手の決定

ある局面における着手可能手は、各自駒が各マスへと移動可能かどうかの判定を行うことによって求まる。ただし、自殺手を避けるために玉将のみ相手駒に取られるマスへの移動はできないものとする。また、王手をかけられた場合は、王手を回避する手以外は無効手として除外し、王手から回避する手のみ有効手とみなす。有効手が存在しない場合は詰み状態となり負けとなる。

2. 評価値の計算

ある局面における評価値は、盤上に存在する駒の種類とその位置により決定される。一般的に将棋は自駒が多いと有利とされるので、自駒に正の値、相手駒に負の値を価値を設定し、その合計値を基準のひとつとする。また、ある局面での着手可能手が多いほど有利とも言われているので、着手可能手数も評価基準としている。ただし、すでに勝負が付いた局面に対しては勝利局面なら評価値を無限大、負け局面なら無限小にする。また同一局面が 3 回出てきた場合は千日手となるので引き分けとし評価値を 0 とする。
表 1 に各駒の評価値、表 2 に着手可能手数の評価値を示す。

表 1 各駒の評価値

駒	歩	銀	金	王
評価値（盤上）	1	4	4	6

表 2 着手可能手数の評価値

着手可能手数	0	1	2	3	4	5	6	7
評価値	0	1	2	3	4	5	6	7

3. 王手の判定

王手の判定は、駒を移動させたときに、その駒が効いているマスを調べ、そのマスに相手の玉将が存在していれば王手と判定する。

4. 勝敗の判定

勝敗の判定は、前述で述べたように王手の判定を行い、王手がかかっている場合は、王手を回避できない手を無効手とみなし着手可能手リストから削除する。その結果、着手可

能手リストに候補手が存在しない場合は、王手を回避することが不可能となるので詰みとなる。

5. 引き分けの判定

千日手の判定は、初期盤面から全ての局面において駒の位置を記憶し、同一局面が3回あった場合は千日手と判断して引き分けとなる。また、先読みを行う際は先読みが行われる以前の盤面と先読み後の盤面とを比較して、千日手と判断された場合にはそれ以降の先読みを中止し、評価値を0にする。

2. ごろごろ動物将棋 AI プログラム

本節では、ごろごろ動物将棋プログラムについて述べる。付録１に、ごろごろ動物将棋のソースを示す。

1. クラス gorogoro

gorogoro クラスは、実行クラスである。ゲーム開始時にユーザーは A チーム、B チームをそれぞれ AI が持つかプレイヤーが持つかを決定し、ゲームを進行していく。

2. クラス Board

Board クラスは、ゲームの盤面を管理するクラスである。表 1 に Board クラスの各メソッドについてまとめる。

表 1 Board クラスのメソッド

メソッド	処理内容
Board()	コンストラクタ
void showBoard()	盤面を表示する
void showPiece()	駒を表示する
String player(int)	各プレイヤーの手番
String com(int)	COM の手番
String ran(int)	COM のランダムの手番
String movePiece(Piece, int, int)	指定した位置に駒を移動させる
void removePiece(int, int)	指定した位置にある駒を取り除く
boolean checkWin(int)	勝負がついたかを調べる
boolean isChecked(int)	現在王手がかかっているかを調べる
boolean isMate(int)	詰みの判定を行う
void createMovableList(int)	候補手を作成する
int value(int)	現在の盤の評価値を表示する
Board nextBoard(NextMove, int)	駒を移動した後の盤面を得る

String name(int)	駒名を返す
void recordBoard()	現在の盤面を記憶する
boolean checkPerpetual()	千日手かどうかを判定する
boolean precheckPerpetual()	先読み時に同じ局面になったかを判定する
void setMaxDepth(int)	先読みの上限を指定する
void resetMoves()	手数をリセット
int winner()	勝者の番号を返す

3. クラス NextMove

NextMove クラスは、駒の移動可能なマスを表すクラスである。移動する駒の種類、マスの座標、評価値をセットおよび返すメソッドがある。表 2 に NextMove クラスの各メソッドについてまとめる。

表 2 NextMove クラスのメソッド

メソッド	処理内容
NextMove()	コンストラクタ
int type	駒の種類を返す
int nextFile()	移動先の X 座標を返す
int nextRank()	移動先の Y 座標を返す
int value	移動した後の盤面の評価値を返す
String toString()	棋譜を返す

4. クラス Piece

Piece クラスは、駒を管理するクラスである。駒の移動できる方向やマス、初期位置や指定したマスへ移動できるかどうか、移動可能リストを返す、移動可能なマスを表示する、クローン生成などのメソッドがある。表 3 に Piece クラスの各メソッドについてまとめる。

表 3 Piece クラスのメソッド

メソッド	処理内容
Piece	コンストラクタ
void setMovableVector()	駒の移動方向をセット
vid setInitialPosition()	駒を盤上の初期位置にセット
void setPosition()	駒を盤上の指定した座標にセット
void setOnBoard()	駒を盤上に置く
void removeFromBoard()	駒を盤上から削除する

boolean isOnBoard()	盤上に駒が存在するか判定する
boolean isThere()	指定した座標に駒が存在するか判定する
int file()	駒の現在のX座標を返す
int rank()	駒の現在のY座標を返す
int type()	駒の種類を返す
String name()	駒名を返す
void move(int, int)	駒を指定した座標へ移動する
boolean movable(int, int)	駒が指定した座標に移動できるか判定する
boolean checkAndMove(int, int)	駒を指定した座標に移動する
ArrayList<NextMove> movableList(int[][])	駒の移動可能な座標を返す
void showMovable(int[][])	駒の移動可能な座標を表示する
Piece clone()	クローンを生成する

4. 実験結果

ごろごろ動物将棋が先手後手どちらが有利であるかを検証するために、本研究で作成したAI同士の対戦を先読み手数4手として100回行ったところ、先手の43勝36敗21引き分けであった。持ち駒と成り駒なしなので千日手が多数発生し、引き分けが多い結果となった。ただし、現状況においてはAI同士の対戦結果から先手が有利ではないかと推測される。

5. 結論・今後の課題

本研究ではごろごろ動物将棋の解析に先駆け、対人戦が可能な成り駒無し、持ち駒無しの簡易版ごろごろ動物将棋アプリケーションを作成した。AI同士の対戦結果より成り駒無し、持ち駒無しの条件では先手有利であることは判明したが、先手必勝であるという根拠は得られなかった。

今後の課題としては、実装されていない持ち駒と成り駒の追加や、アプリケーションの高速化、適切な評価値の探索、詰み局面の列挙をすることで完全解析することが挙げられる。しかし、完全解析済みの動物将棋の総局数が1,567,925,964通りである[13]のに対して、ごろごろ動物将棋の総局数は1.0×10^{22}通りあるので、完全解析は困難であると予測される。しかし、より強いプログラムを作るということは可能である。より強くするための戦略としては、駒の評価値を自玉近くの自駒と相手駒の数を比較して、相手駒の方が多いなら自玉を守るために駒を自玉に近付ける手に高評価を与え、自駒の方が多いなら相手玉を攻めるために駒を敵陣へと進める手を高評価にするまた、本将棋において銀将は攻め駒、金将は守り駒として使われることが多いので、金は自玉近くなら高評価に、銀将は相手玉近くなら高評価にする、定跡・対戦データベースを追加する、評価値の高い駒を優先的に取るようにするなどが考えられる。

参考文献

[1] ごろごろどうぶつしょうぎ、幻冬舎エデュケーション （2012）
http://www.gentosha-edu.co.jp/products/post-132.html

[2] Michael Khodarkovsky, Leonid Shamkovich、
人間対機械ー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)

[3] 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)

[4] MicaelBuro、Kenta Tominaga vs. Logistello、2012、
https://skatgame.net/mburo/iwec.html

[5] Janos Wagner、Istvan Virag、Solving renjyu、
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 Kisimoto、
Martin Muller、Robert Lake、Paul Lu、Steve Suphen、Checkers is solved、
Science Vol.317、No.5844、pp.1518-1522(2007)、
http://www.siencemag.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/fbnal.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、Jos W.H.M.Uiterwijk、
Solving Go on Small Boards、IcCa Journal、Vol.26、No.2、pp92-107(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、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://doubutushogi.net/

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

[14] IBM100-DeepBule、IBM、1997
http://www-03.ibm.com/ibm/history/ibm100/us/en/icons/deepblue

[15] 伊藤英紀、A 級リーグ指し手 1 号、2014、http://aleag.cocolog-nifty.com/

[16] Michael Buro、LOGISTELLO、2002、http://skatgame.net/mburo/log.html

[17] 電王戦、日本将棋連盟、2014、http://www.shogi.or.jp/kisen/denou/

[18] 一丸貴則、コンピュータ将棋開発中、2014、
http://www.computer-shogi.org/wcsc21/appeal/tsutsukana/WCSC21_tsutsukana_20110327.pdf

[19] 電王戦リベンジマッチ、日本将棋連盟、2014、
http://www.shogi.or.jp/topics/2014/01/post-899.html

謝辞
本研究では石水先生や、研究室の皆さんには大変お世話になりました。
ありがとうございます。

付録

以下に本研究で作成したごろごろ動物将棋プログラムのソースを示す。

クラス gorogoro

```
package gorogoro;

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

import gorogoro.Board;

public class gorogoro {
	//A チーム　先手の駒　盤上手前
	final static int LION = 1; //ライオン
	final static int DOG_1 = 2; //犬_1
	final static int DOG_2 = 3; //犬_2
	final static int NEKO_1 = 4; //猫_1
	final static int NEKO_2 = 5; //猫_2
	final static int HIYOKO_1 = 6; //ひよこ_1
	final static int HIYOKO_2 = 7; //ひよこ_2
	final static int HIYOKO_3 = 8; //ひよこ_3
	/*
	final static int CAT_1 = 9; //成り猫_1
	final static int CAT_2 = 10; //成り猫_2
	final static int BIRD_1 = 11; //成りひよこ_1
	final static int BIRD_2 = 12; //成りひよこ_2
	final static int BIRD_3 = 13; //成りひよこ_3
*/

	//B チーム　後手の駒　盤上奥
	final static int E_LION = -1; //ライオン
	final static int E_DOG_1 = -2; //犬_1
	final static int E_DOG_2 = -3; //犬_2
	final static int E_NEKO_1 = -4; //猫_1
	final static int E_NEKO_2 = -5; //猫_2
	final static int E_HIYOKO_1 = -6; //ひよこ_1
	final static int E_HIYOKO_2 = -7; //ひよこ_2
	final static int E_HIYOKO_3 = -8; //ひよこ_3
	/*
```

```
	final static int E_CAT_1 = -9; //成り猫_1
	final static int E_CAT_2 = -10; //成り猫_2
	final static int E_BIRD_1 = -11; //成りひよこ_1
	final static int E_BIRD_2 = -12; //成りひよこ_12
	final static int E_BIRD_3 = -13; //成りひよこ_13
*/

	final static int EMPTY = 0; //空白
	final static int BORDER = Integer.MAX_VALUE; // 盤外
	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); //対人戦の入力確認
		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 ("gorogoroScore.txt");
			fp = new PrintWriter (pass);
		} catch (IOException e) {
			System.err.println (e);
		}

		//対局を決める
		System.out.print ("A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 ("B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); // A 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); // B 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();
     }
}
```

クラス Board

```
package gorogoro;

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

import gorogoro.NextMove;
import gorogoro.Piece;

public class Board {
     //A チーム　先手の駒　盤上手前
     final static int LION = 1; //ライオン
```

```
        final static int DOG_1 = 2; //犬_1
        final static int DOG_2 = 3; //犬_2
        final static int NEKO_1 = 4; //猫_1
        final static int NEKO_2 = 5; //猫_2
        final static int HIYOKO_1 = 6; //ひよこ_1
        final static int HIYOKO_2 = 7; //ひよこ_2
        final static int HIYOKO_3 = 8; //ひよこ_3
        /*
        final static int CAT_1 = 9; //成り猫_1
        final static int CAT_2 = 10; //成り猫_2
        final static int BIRD_1 = 11; //成りひよこ_1
        final static int BIRD_2 = 12; //成りひよこ_2
        final static int BIRD_3 = 13; //成りひよこ_3
        */

        //Bチーム　後手の駒　盤上奥
        final static int E_LION = -1; //ライオン
        final static int E_DOG_1 = -2; //犬_1
        final static int E_DOG_2 = -3; //犬_2
        final static int E_NEKO_1 = -4; //猫_1
        final static int E_NEKO_2 = -5; //猫_2
        final static int E_HIYOKO_1 = -6; //ひよこ_1
        final static int E_HIYOKO_2 = -7; //ひよこ_2
        final static int E_HIYOKO_3 = -8; //ひよこ_3
        /*
        final static int E_CAT_1 = -9; //成り猫_1
        final static int E_CAT_2 = -10; //成り猫_2
        final static int E_BIRD_1 = -11; //成りひよこ_1
        final static int E_BIRD_2 = -12; //成りひよこ_12
        final static int E_BIRD_3 = -13; //成りひよこ_13
*/

        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,
BORDER},
         {BORDER, E_NEKO_1, E_DOG_1,    E_LION,     E_DOG_2,    E_NEKO_2,
BORDER},
         {BORDER, EMPTY,    EMPTY,      EMPTY,      EMPTY,      EMPTY,
BORDER},
         {BORDER, EMPTY,    E_HIYOKO_1, E_HIYOKO_2, E_HIYOKO_3, EMPTY,
BORDER},
         {BORDER, EMPTY,    HIYOKO_1,   HIYOKO_2,   HIYOKO_3,   EMPTY,
BORDER},
         {BORDER, EMPTY,    EMPTY,      EMPTY,      EMPTY,      EMPTY,
BORDER},
         {BORDER, NEKO_1,   DOG_1,      LION,       DOG_2,      NEKO_2,
BORDER},
         {BORDER, BORDER,   BORDER,     BORDER,     BORDER,     BORDER,
BORDER}};

      int nextFile; // 移動先のX座標
      int nextRank; // 移動先のY座標

      Piece lion, dog_1, dog_2, neko_1, neko_2, hiyoko_1, hiyoko_2, hiyoko_3,
/*cat_1, cat_2, bird_1, bird_2, bird_3,*/
                  e_lion, e_dog_1, e_dog_2, e_neko_1, e_neko_2, e_hiyoko_1,
e_hiyoko_2, e_hiyoko_3/*, e_cat_1, e_cat_2, e_bird_1, e_bird_2, e_bird_3*/; //
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

      /*
       * ＠３ならばスラスラ
       * ＠４ならば１手に六秒前後
       */
      int maxDepth = 4; // 先読みする手数の上限;
      boolean doResign = false; // 負けが確定したときにCOMが投了するか
      static int moves = 0; // 手数
      static int lookAheadMoves = 0; // 先読み手数
      final static int maxMove = 500; // 手数の上限
      static int[][][] boardRecord = new int [maxMove][6][5]; // 盤面記録用
```

```
int winner = 0; // 勝者 1:先手勝ち -1:後手勝ち 0:引き分け

/**
 * コンストラクタ
 * 盤上に駒を初期設定で生成
 */
public Board () {
	lion		= new Piece (LION);
	dog_1		= new Piece (DOG_1);
	dog_2		= new Piece (DOG_2);
	neko_1		= new Piece (NEKO_1);
	neko_2		= new Piece (NEKO_2);
	hiyoko_1	= new Piece (HIYOKO_1);
	hiyoko_2	= new Piece (HIYOKO_2);
	hiyoko_3	= new Piece (HIYOKO_3);
	/*
	cat_1		= new Piece (CAT_1);
	cat_2		= new Piece (CAT_2);
	bird_1		= new Piece (BIRD_1);
	bird_2		= new Piece (BIRD_2);
	bird_3		= new Piece (BIRD_3);
	*/

	e_lion		= new Piece (E_LION);
	e_dog_1		= new Piece (E_DOG_1);
	e_dog_2		= new Piece (E_DOG_2);
	e_neko_1	= new Piece (E_NEKO_1);
	e_neko_2	= new Piece (E_NEKO_2);
	e_hiyoko_1	= new Piece (E_HIYOKO_1);
	e_hiyoko_2	= new Piece (E_HIYOKO_2);
	e_hiyoko_3	= new Piece (E_HIYOKO_3);
	/*
	e_cat_1		= new Piece (E_CAT_1);
	e_cat_2		= new Piece (E_CAT_2);
	e_bird_1	= new Piece (E_BIRD_1);
	e_bird_2	= new Piece (E_BIRD_2);
	e_bird_3	= new Piece (E_BIRD_3);
	*/
}

/**
```

```
 * 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 <= 6; ++r)
            for (int f = 1; f <= 5; ++f)
                  this.board[r][f] = board[r][f];
      for (int r = 1 ; r <= 6; ++r) {
            for (int f = 1; f <= 5; ++f) {
                  switch (board [r][f]) {
                  case LION :
                        lion = new Piece (LION, f, r);
                        break;
                  case DOG_1 :
                         dog_1 = new Piece (DOG_1, f, r);
                        break;
                  case DOG_2 :
                        dog_2 = new Piece (DOG_2, f, r);
                        break;
                  case NEKO_1 :
                        neko_1 = new Piece (NEKO_1, f, r);
                        break;
                  case NEKO_2 :
                        neko_2 = new Piece (NEKO_2, f, r);
                        break;
                  case HIYOKO_1 :
                        hiyoko_1 = new Piece (HIYOKO_1, f, r);
                        break;
                  case HIYOKO_2 :
                        hiyoko_2 = new Piece (HIYOKO_2, f, r);
                        break;
                  case HIYOKO_3 :
                        hiyoko_3 = new Piece (HIYOKO_3, f, r);
                        break;
                        /*
                  case CAT_1 :
                        cat_1 = new Piece (CAT_1, f, r);
                        break;
                  case CAT_2 :
                        cat_2 = new Piece (CAT_2, f, r);
```

```
        break;
case BIRD_1 :
        bird_1 = new Piece (BIRD_1, f, r);
        break;
case BIRD_2 :
        bird_2 = new Piece (BIRD_2, f, r);
        break;
case BIRD_3 :
        bird_3 = new Piece (BIRD_3, f, r);
        break;
        */

case E_LION :
        e_lion = new Piece (E_LION, f, r);
        break;
case E_DOG_1 :
         e_dog_1 = new Piece (E_DOG_1, f, r);
        break;
case E_DOG_2 :
        e_dog_2 = new Piece (E_DOG_2, f, r);
        break;
case E_NEKO_1 :
        e_neko_1 = new Piece (E_NEKO_1, f, r);
        break;
case E_NEKO_2 :
        e_neko_2 = new Piece (E_NEKO_2, f, r);
        break;
case E_HIYOKO_1 :
        e_hiyoko_1 = new Piece (E_HIYOKO_1, f, r);
        break;
case E_HIYOKO_2 :
        e_hiyoko_2 = new Piece (E_HIYOKO_2, f, r);
        break;
case E_HIYOKO_3 :
        e_hiyoko_3 = new Piece (E_HIYOKO_3, f, r);
        break;
        /*
case E_CAT_1 :
        e_cat_1 = new Piece (E_CAT_1, f, r);
        break;
case E_CAT_2 :
```

```
                        e_cat_2 = new Piece (E_CAT_2, f, r);
                        break;
                  case E_BIRD_1 :
                        e_bird_1 = new Piece (E_BIRD_1, f, r);
                        break;
                  case E_BIRD_2 :
                        e_bird_2 = new Piece (E_BIRD_2, f, r);
                        break;
                  case E_BIRD_3 :
                        e_bird_3 = new Piece (E_BIRD_3, f, r);
                        break;
                        */
                  }
            }
      }
}

/**
 * 盤を表示
 */
public void showBoard () {
      System.out.println ("    １２３４５");
      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("六");
                break;
            case 7 :
                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 LION :
                return "ら";
        case DOG_1 :
                return "い";
        case DOG_2 :
                return "い";
        case NEKO_1 :
                return "ね";
        case NEKO_2 :
                return "ね";
        case HIYOKO_1 :
                return "ひ";
        case HIYOKO_2 :
                return "ひ";
        case HIYOKO_3 :
                return "ひ";
                /*
```

```
case CAT_1 :
      return "き";
case CAT_2 :
      return "き";
case BIRD_1 :
      return "と";
case BIRD_2 :
      return "と";
case BIRD_3 :
      return "と";
      */
case E_LION :
      return "ラ";
case E_DOG_1 :
      return "イ";
case E_DOG_2 :
      return "イ";
case E_NEKO_1 :
      return "ネ";
case E_NEKO_2 :
      return "ネ";
case E_HIYOKO_1 :
      return "ヒ";
case E_HIYOKO_2 :
      return "ヒ";
case E_HIYOKO_3 :
      return "ヒ";
      /*
case E_CAT_1 :
      return "キ";
case E_CAT_2 :
      return "キ";
case E_BIRD_1 :
      return "ト";
case E_BIRD_2 :
      return "ト";
case E_BIRD_3 :
      return "ト";
      */
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 ("A チームの番です");
                //System.out.print ("進める駒(L,D1,D2,N1,N2,H1,H2,H3)を選
んでください(R=投了):");
                System.out.print("進める駒(");

                if(lion != null)
                    System.out.print("L ");
                if(dog_1 != null)
                    System.out.print("D1 ");
                if(dog_2 != null)
                    System.out.print("D2 ");
                if(neko_1 != null)
                    System.out.print("N1 ");
                if(neko_2 != null)
                    System.out.print("N2 ");
                if(hiyoko_1 != null)
                    System.out.print("H1 ");
```

```
                    if(hiyoko_2 != null)
                        System.out.print("H2 ");
                    if(hiyoko_3 != null)
                        System.out.print("H3 ");

                    System.out.print(")を選んでください(R=投了):");
                } else {
                    System.out.println ("Bチームの番です");
                    //System.out.print ("進める駒
(EL,ED1,ED2,EN1,EN2,EH1,EH2,EH3)を選んでください(R=投了):");
                    System.out.print("進める駒(");

                    if(e_lion != null)
                        System.out.print("EL ");
                    if(e_dog_1 != null)
                        System.out.print("ED1 ");
                    if(e_dog_2 != null)
                        System.out.print("ED2 ");
                    if(e_neko_1 != null)
                        System.out.print("EN1 ");
                    if(e_neko_2 != null)
                        System.out.print("EN2 ");
                    if(e_hiyoko_1 != null)
                        System.out.print("EH1 ");
                    if(e_hiyoko_2 != null)
                        System.out.print("EH2 ");
                    if(e_hiyoko_3 != null)
                        System.out.print("EH3 ");

                    System.out.print(")を選んでください(R=投了):");
                }
                inputPiece = keyBoardScanner.next();
                if (inputPiece.equals ("L") || inputPiece.equals ("l")) {
                    piece = lion;
                    type = LION;
                } else if (inputPiece.equals ("D1") || inputPiece.equals
("d1")) {
                    piece = dog_1;
                    type = DOG_1;
                } else if (inputPiece.equals ("D2") || inputPiece.equals
("d2")) {
```

```
                    piece = dog_2;
                    type = DOG_2;
                } else if (inputPiece.equals ("N1") || inputPiece.equals
("n1")) {
                    piece = neko_1;
                    type = NEKO_1;
                } else if (inputPiece.equals ("N2") || inputPiece.equals
("n2")) {
                    piece = neko_2;
                    type = NEKO_2;
                } else if (inputPiece.equals ("H1") || inputPiece.equals
("h1")) {
                    piece = hiyoko_1;
                    type = HIYOKO_1;
                } else if (inputPiece.equals ("H2") || inputPiece.equals
("h2")) {
                    piece = hiyoko_2;
                    type = HIYOKO_2;
                } else if (inputPiece.equals ("H3") || inputPiece.equals
("h3")) {
                    piece = hiyoko_3;
                    type = HIYOKO_3;
                }
                /*else if (inputPiece.equals ("C1") || inputPiece.equals
("c1")) {
                    piece = cat_1;
                    type = CAT_1;
                } else if (inputPiece.equals ("C2") || inputPiece.equals
("c2")) {
                    piece = cat_1;
                    type = CAT_2;
                } else if (inputPiece.equals ("B1") || inputPiece.equals
("b1")) {
                    piece = bird_1;
                    type = BIRD_1;
                } else if (inputPiece.equals ("B2") || inputPiece.equals
("b2")) {
                    piece = bird_2;
                    type = BIRD_2;
                } else if (inputPiece.equals ("B3") || inputPiece.equals
("b3")) {
```

```
                    piece = bird_3;
                    type = BIRD_3;
                }
                */
                else if (inputPiece.equals ("EL") || inputPiece.equals ("el"))
{
                    piece = e_lion;
                    type = E_LION;
                } else if (inputPiece.equals ("ED1") || inputPiece.equals
("ed1")) {
                    piece = e_dog_1;
                    type = E_DOG_1;
                } else if (inputPiece.equals ("ED2") || inputPiece.equals
("ed2")) {
                    piece = e_dog_2;
                    type = E_DOG_2;
                } else if (inputPiece.equals ("EN1") || inputPiece.equals
("en1")) {
                    piece = e_neko_1;
                    type = E_NEKO_1;
                } else if (inputPiece.equals ("EN2") || inputPiece.equals
("en2")) {
                    piece = e_neko_2;
                    type = E_NEKO_2;
                } else if (inputPiece.equals ("EH1") || inputPiece.equals
("eh1")) {
                    piece = e_hiyoko_1;
                    type = E_HIYOKO_1;
                } else if (inputPiece.equals ("EH2") || inputPiece.equals
("eh2")) {
                    piece = e_hiyoko_2;
                    type = E_HIYOKO_2;
                } else if (inputPiece.equals ("EH3") || inputPiece.equals
("eh3")) {
                    piece = e_hiyoko_3;
                    type = E_HIYOKO_3;
                }
                /*else if (inputPiece.equals ("EC1") || inputPiece.equals
("ec1")) {
                    piece = e_cat_1;
                    type = E_CAT_1;
```

```
                } else if (inputPiece.equals ("EC2") || inputPiece.equals
("ec2")) {
                        piece = e_cat_2;
                        type = E_CAT_2;
                } else if (inputPiece.equals ("EB1") || inputPiece.equals
("eb1")) {
                        piece = e_bird_1;
                        type = E_BIRD_1;
                }  else if (inputPiece.equals ("EB2") || inputPiece.equals
("eb2")) {
                        piece = e_bird_2;
                        type = E_BIRD_2;
                } else if (inputPiece.equals ("EB3") || inputPiece.equals
("eb3")) {
                        piece = e_bird_3;
                        type = E_BIRD_3;
                }
                */
                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 ("L,D1,D2,N1,N2,H1,H2,H3 から
選んでください");
                                if(lion != null)
                                        System.out.print("L ");
                                if(dog_1 != null)
                                        System.out.print("D1 ");
                                if(dog_2 != null)
                                        System.out.print("D2 ");
                                if(neko_1 != null)
```

```
                                    System.out.print("N1 ");
                               if(neko_2 != null)
                                    System.out.print("N2 ");
                               if(hiyoko_1 != null)
                                    System.out.print("H1 ");
                               if(hiyoko_2 != null)
                                    System.out.print("H2 ");
                               if(hiyoko_3 != null)
                                    System.out.print("H3 ");
                               System.out.println ("から選んでください");
                          } else {
                               //System.out.println
("EL,ED1,ED2,EN1,EN2,EH1,EH2,EH3 から選んでください");
                               if(e_lion != null)
                                    System.out.print("EL ");
                               if(e_dog_1 != null)
                                    System.out.print("ED1 ");
                               if(e_dog_2 != null)
                                    System.out.print("ED2 ");
                               if(e_neko_1 != null)
                                    System.out.print("EN1 ");
                               if(e_neko_2 != null)
                                    System.out.print("EN2 ");
                               if(e_hiyoko_1 != null)
                                    System.out.print("EH1 ");
                               if(e_hiyoko_2 != null)
                                    System.out.print("EH2 ");
                               if(e_hiyoko_3 != null)
                                    System.out.print("EH3 ");
                               System.out.println ("から選んでください");
                          }
                          continue; // 選び直し
                     }

                     if (playerNum == 0 && !(type == LION || type == DOG_1 || type
== DOG_2 || type == NEKO_1 || type == NEKO_2 || type == HIYOKO_1 || type ==
HIYOKO_2 || type == HIYOKO_3
                                                  /* || type == CAT_1 || type
== CAT_2 || type == BIRD_1 || type == BIRD_2 || type == BIRD_3*/)) {
                          //System.out.println ("L,D1,D2,N1,N2,H1,H2,H3 から選んでく
ださい");
```

```
                        if(lion != null)
                              System.out.print("L ");
                        if(dog_1 != null)
                              System.out.print("D1 ");
                        if(dog_2 != null)
                              System.out.print("D2 ");
                        if(neko_1 != null)
                              System.out.print("N1 ");
                        if(neko_2 != null)
                              System.out.print("N2 ");
                        if(hiyoko_1 != null)
                              System.out.print("H1 ");
                        if(hiyoko_2 != null)
                              System.out.print("H2 ");
                        if(hiyoko_3 != null)
                              System.out.print("H3 ");
                        System.out.println ("から選んでください");
                        continue; // 選び直し
                  } else if (playerNum == 1 && !(type == E_LION || type ==
E_DOG_1 || type == E_DOG_2 || type == E_NEKO_1 || type == E_NEKO_2 || type ==
E_HIYOKO_1 || type == E_HIYOKO_2 || type == E_HIYOKO_3
                                                                  /*|| type ==
E_CAT_1 || type == E_CAT_2 || type == E_BIRD_1 || type == E_BIRD_2 || type ==
E_BIRD_3*/)) {
                        //System.out.println ("EL,ED1,ED2,EN1,EN2,EH1,EH2,EH3 か
ら選んでください");
                        if(e_lion != null)
                              System.out.print("EL ");
                        if(e_dog_1 != null)
                              System.out.print("ED1 ");
                        if(e_dog_2 != null)
                              System.out.print("ED2 ");
                        if(e_neko_1 != null)
                              System.out.print("EN1 ");
                        if(e_neko_2 != null)
                              System.out.print("EN2 ");
                        if(e_hiyoko_1 != null)
                              System.out.print("EH1 ");
                        if(e_hiyoko_2 != null)
                              System.out.print("EH2 ");
                        if(e_hiyoko_3 != null)
```

```
                        System.out.print("EH3 ");
                    System.out.println ("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(1～5):");
                nextFile = keyBoardScanner.nextInt();
                System.out.print ("y座標(1～6):");
                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
        }
        String score = movePiece (piece, type, nextFile, nextRank); // 駒を移
動させる
        System.out.println (moves + ":" + score + "[" + value(playerNum) +
"]");
        ++moves;
        return score;
    }

    /**
     * COMの手番
     * maxDepth で指定した手数を先読みして最も良い手を指す
     * @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 = 0;                  // 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 (movableList.size() == 0) { // 候補手が存在しない＝詰み
                System.out.println ("投了します");
```

```
                resign = true;
                if (isChessStyleScore) {
                        return (playerNum == 0) ? "1-0" : "0-1";
                } else {
                        return "投了";
                }
        } else if (movableList.size() == 1) { // 候補手が1つしか無い場合は評価値を求めない
                bestMove = movableList.get(0);
        } else {
                if (playerNum == 0) { // A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;
                        bestMove = movableList.get(0);

                        ++lookAheadMoves; // 先読みのために手数を1増やす
                        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
                                }
                        }
                        --lookAheadMoves; // 先読みが終了したので手数を1戻す
                        if (bestValue == Integer.MIN_VALUE && doResign) {  // 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 { // B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;
                  bestMove = movableList.get(0);
                  ++lookAheadMoves; // 先読みのために手数を1増やす
                  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
                              --lookAheadMoves; // 先読みが終了したので手数を
1戻す
                              break;                                // ルー
プ脱出
                        }
                  }
                  --lookAheadMoves; // 先読みが終了したので手数を1戻す
                  if (bestValue == Integer.MAX_VALUE && doResign) {  // 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 LION     : piece = lion;     break;
case DOG_1    : piece = dog_1;    break;
case DOG_2    : piece = dog_2;    break;
case NEKO_1   : piece = neko_1;   break;
case NEKO_2   : piece = neko_2;   break;
case HIYOKO_1 : piece = hiyoko_1; break;
case HIYOKO_2 : piece = hiyoko_2; break;
case HIYOKO_3 : piece = hiyoko_3; break;
/*
case CAT_1    : piece = cat_1;    break;
case CAT_2    : piece = cat_2;    break;
case BIRD_1   : piece = bird_1;   break;
case BIRD_2   : piece = bird_2;   break;
case BIRD_3   : piece = bird_3;   break;
*/

case E_LION     : piece = e_lion;     break;
case E_DOG_1    : piece = e_dog_1;    break;
case E_DOG_2    : piece = e_dog_2;    break;
case E_NEKO_1   : piece = e_neko_1;   break;
case E_NEKO_2   : piece = e_neko_2;   break;
case E_HIYOKO_1 : piece = e_hiyoko_1; break;
case E_HIYOKO_2 : piece = e_hiyoko_2; break;
case E_HIYOKO_3 : piece = e_hiyoko_3; break;
/*
case E_CAT_1    : piece = e_cat_1;    break;
case E_CAT_2    : piece = e_cat_2;    break;
```

```
            case E_BIRD_1   : piece = e_bird_1;   break;
            case E_BIRD_2   : piece = e_bird_2;   break;
            case E_BIRD_3   : piece = e_bird_3;   break;
            */
            }
            String score = movePiece (piece, type, nextFile, nextRank); // 駒を移
動させる
            System.out.println (moves + ":" + score + "[" + bestValue + "]");
            ++moves;
            return score;
      }

      /**
       * COMの手番
       * ランダムに手を指す
       * @paran int playerNum プレイヤー番号
       * @return 指した手の棋譜
       */
      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

            NextMove ranMove = null;            // ランダム手
            boolean isMateToOne = false;        // 1手詰めか
            int nextPlayerNum;                  // 次の手番プレイヤー

            if (movableList.size() == 0) { // 候補手が存在しない＝詰み
                  System.out.println ("投了します");
                  resign = true;
                  if (isChessStyleScore) {
                        return (playerNum == 0) ? "1-0" : "0-1";
                  } else {
                        return "投了";
                  }
            } else if (movableList.size() == 1) { // 候補手が1つしか無い場合は評価
値を求めない
                  ranMove = movableList.get(0);
            } else {
                  if (playerNum == 0) { // A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
                    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
                        //nextBoard.createMovableList (playerNum);    // 候
補手を生成
                        int value = nextBoard.value (nextPlayerNum, 0);//
先読み無しで盤面の評価値を計算
                        if (value == Integer.MAX_VALUE) {        // 1 手詰め
の手を発見した
                            ranMove = nextMove;                    // 1 手
詰めの手を記憶
                            isMateToOne = true;
                            break;                  // ループ脱出
                        }
                    }
                } else { // B 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
                    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
                        //nextBoard.createMovableList (playerNum);    // 候
補手を生成
                        int value = nextBoard.value (nextPlayerNum, 0);//
先読み無しで盤面の評価値を計算
                        if (value == Integer.MIN_VALUE) {        // 1 手詰め
の手を発見した
                            ranMove = nextMove;                    // 1 手
詰めの手を記憶
                            isMateToOne = true;
                            break;                  // ループ脱出
                        }
                    }
                }
                if (!isMateToOne) { // 1 手詰めの手は無かった
```

```
                Random rnd = new Random();
                int ran = rnd.nextInt (movableList.size()); // 候補手数までの乱数を生成
                ranMove = movableList.get (ran);
            }
        }
        type = ranMove.type();          // 移動する駒の種類
        nextFile = ranMove.nextFile(); // 移動先のX座標
        nextRank = ranMove.nextRank(); // 移動先のY座標

        switch (type) {
        case LION     : piece = lion;     break;
        case DOG_1    : piece = dog_1;    break;
        case DOG_2    : piece = dog_2;    break;
        case NEKO_1   : piece = neko_1;   break;
        case NEKO_2   : piece = neko_2;   break;
        case HIYOKO_1 : piece = hiyoko_1; break;
        case HIYOKO_2 : piece = hiyoko_2; break;
        case HIYOKO_3 : piece = hiyoko_3; break;
        /*
        case CAT_1    : piece = cat_1;    break;
        case CAT_2    : piece = cat_2;    break;
        case BIRD_1   : piece = bird_1;   break;
        case BIRD_2   : piece = bird_2;   break;
        case BIRD_3   : piece = bird_3;   break;
        */

        case E_LION     : piece = e_lion;     break;
        case E_DOG_1    : piece = e_dog_1;    break;
        case E_DOG_2    : piece = e_dog_2;    break;
        case E_NEKO_1   : piece = e_neko_1;   break;
        case E_NEKO_2   : piece = e_neko_2;   break;
        case E_HIYOKO_1 : piece = e_hiyoko_1; break;
        case E_HIYOKO_2 : piece = e_hiyoko_2; break;
        case E_HIYOKO_3 : piece = e_hiyoko_3; break;
        /*
        case E_CAT_1    : piece = e_cat_1;    break;
        case E_CAT_2    : piece = e_cat_2;    break;
        case E_BIRD_1   : piece = e_bird_1;   break;
        case E_BIRD_2   : piece = e_bird_2;   break;
        case E_BIRD_3   : piece = e_bird_3;   break;
```

```
            */
            }
            String score = movePiece (piece, type, nextFile, nextRank); // 駒を移
動させる
            System.out.println (moves + ":" + score + "[?]");
            ++moves;
            return score;
      }


      /**
       * 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 LION : score = "L"; break;
                  case DOG_1 : score = "D1"; break;
                  case DOG_2 : score = "D2"; break;
                  case NEKO_1 : score = "N1"; break;
                  case NEKO_2 : score = "N2"; break;
                  case HIYOKO_1 : score = "H1"; break;
                  case HIYOKO_2 : score = "H2"; break;
                  case HIYOKO_3 : score = "H3"; break;
                  /*
                  case CAT_1 : score = "C1"; break;
                  case CAT_2 : score = "C2"; break;
                  case BIRD_1 : score = "B1"; break;
                  case BIRD_2 : score = "B2"; break;
                  case BIRD_3 : score = "B3"; break;
                  */

                  case E_LION : score ="EL"; break;
                  case E_DOG_1 : score = "ED1"; break;
```

```
        case E_DOG_2 : score = "ED2"; break;
        case E_NEKO_1 : score = "EN1"; break;
        case E_NEKO_2 : score = "EN2"; break;
        case E_HIYOKO_1 : score = "EH1"; break;
        case E_HIYOKO_2 : score ="EH2"; break;
        case E_HIYOKO_3 : score = "EH3"; break;
        /*
        case E_CAT_1 : score = "EC1"; break;
        case E_CAT_2 : score = "EC2"; break;
        case E_BIRD_1 : score = "EB1"; break;
        case E_BIRD_2 : score = "EB2"; break;
        case E_BIRD_3 : score = "EB3"; 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;
        case 4 :  score += "d"; break;
        case 5 :  score += "e"; 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;
        case 6 :  score += "6 "; 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;
        case 4 :  score = "４"; break;
        case 5 :  score = "５"; 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;
case 6 :  score += "六"; break;
default : score += "？" ; break;
}
switch (type) {
case LION       : score += "ら   "; break; //ライオン
case DOG_1      : score += "い_1  "; break; //犬_1
case DOG_2      : score += "い_2  "; break; //犬_2
case NEKO_1     : score += "ね_1  "; break; //猫_1
case NEKO_2     : score += "ね_2  "; break; //猫_2
case HIYOKO_1   : score += "ひ_1  "; break; //ひよこ_1
case HIYOKO_2   : score += "ひ_2  "; break; //ひよこ_2
case HIYOKO_3   : score += "ひ_3  "; break; //ひよこ_3
/*@
case CAT_1      : score += "き_1  "; break; //成り猫_1
case CAT_2      : score += "き_2  "; break; //成り猫_2
case BIRD_1     : score += "と_1  "; break; //成りひよこ_1
case BIRD_2     : score += "と_2  "; break; //成りひよこ_2
case BIRD_3     : score += "と_3  "; break; //成りひよこ_3
*/

case E_LION     : score += "ラ   "; break; //ライオン
case E_DOG_1    : score += "イ_1  "; break; //犬_1
case E_DOG_2    : score += "イ_2  "; break; //犬_2
case E_NEKO_1   : score += "ネ_1  "; break; //猫_1
case E_NEKO_2   : score += "ネ_2  "; break; //猫_2
case E_HIYOKO_1 : score += "ヒ_1  "; break; //ひよこ_1
case E_HIYOKO_2 : score += "ヒ_2  "; break; //ひよこ_2
case E_HIYOKO_3 : score += "ヒ_3  "; break; //ひよこ_3
/*
case E_CAT_1    : score += "キ_1  "; break; //成り猫_1
case E_CAT_2    : score += "キ_2  "; break; //成り猫_2
```

```
                case E_BIRD_1   : score += "ト_1  "; break; //成りひよこ_1
                case E_BIRD_2   : score += "ト_2  "; break; //成りひよこ_2
                case E_BIRD_3   : score += "ト_3  "; break; //成りひよこ_3
                */
                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

        //ひよこの成り

        recordBoard(); // 移動後の盤面を記録
        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 LION :                    // 移動先にライオン
                lion = null;             // ライオンを取り除く
                break;
        case DOG_1 :                 // 移動先に犬_1
                dog_1 = null;            // 犬_1を取り除く
                break;
        case DOG_2 :                 // 移動先に犬_2
                dog_2 = null;            // 犬_2を取り除く
                break;
        case NEKO_1 :                        // 移動先に猫_1
                neko_1 = null;               // 猫_1を取り除く
```

```
        break;
case NEKO_2 :                       // 移動先に猫_2
        neko_2 = null;              // 猫_2を取り除く
        break;
case HIYOKO_1 :                     // 移動先にひよこ_1
        hiyoko_1 = null;          // ひよこ_1を取り除く
        break;
case HIYOKO_2 :                     // 移動先にひよこ_2
        hiyoko_2 = null;          // ひよこ_2を取り除く
        break;
case HIYOKO_3 :                     // 移動先にひよこ_3
        hiyoko_3 = null;          // ひよこ_3を取り除く
        break;
        /*
case CAT_1 :                        // 移動先に成り猫_1
         cat_1 = null;              // 成り猫_1を取り除く
         break;
case CAT_2 :                        // 移動先に成り猫_2
        cat_2 = null;               // 成り猫_2を取り除く
        break;
case BIRD_1 :                       // 移動先に成りひよこ_1
        bird_1 = null;              // 成りひよこ_1を取り除く
        break;
case BIRD_2 :                       // 移動先に成りひよこ_2
        bird_2 = null;              // 成りひよこ_2を取り除く
        break;
case BIRD_3 :                       // 移動先に成りひよこ_3
        bird_3 = null;              // 成りひよこ_3を取り除く
        break;
        */

case E_LION :                     // 移動先にE-ライオン
        e_lion = null;              // E-ライオンを取り除く
        break;
case E_DOG_1 :                  // 移動先にE-犬_1
        e_dog_1 = null;           // E-犬_1を取り除く
        break;
case E_DOG_2 :                  // 移動先にE=犬_2
        e_dog_2 = null;           // E-犬_2を取り除く
        break;
```

```
        case E_NEKO_1 :                         // 移動先にE-猫_1
              e_neko_1 = null;                  // E-猫_1を取り除く
              break;
        case E_NEKO_2 :                         // 移動先にE-猫_2
              e_neko_2 = null;                  // E-猫_2を取り除く
              break;
        case E_HIYOKO_1 :                       // 移動先にE-ひよこ_1
              e_hiyoko_1 = null;                // E-ひよこ_1を取り除く
              break;
        case E_HIYOKO_2 :                       // 移動先にE-ひよこ_2
              e_hiyoko_2 = null;                // E-ひよこ_2を取り除く
              break;
        case E_HIYOKO_3 :                       // 移動先にE-ひよこ_3
              e_hiyoko_3 = null;                // E-ひよこ_3を取り除く
              break;
              /*
        case E_CAT_1 :                          // 移動先にE-成り猫_1
               e_cat_1 = null;                  // E_成り猫_1を取り除く
               break;
        case E_CAT_2 :                          // 移動先にE-成り猫_2
              e_cat_2 = null;                   // E-成り猫_2を取り除く
              break;
        case E_BIRD_1 :                         // 移動先にE-成りひよこ_1
              e_bird_1 = null;                  // E-成りひよこ_1を取り除く
              break;
        case E_BIRD_2 :                         // 移動先にE-成りひよこ_2
              e_bird_2 = null;                  // E-成りひよこ_2を取り除く
              break;
        case E_BIRD_3 :                         // 移動先にE-成りひよこ_3
              e_bird_3 = null;                  // E-成りひよこ_3を取り除く
              break;
              */
        }
}

/**
 * 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) {                    // Aチームの手番
            if (e_lion == null) {              // E-ライオンを取った
                  System.out.println ("Aチームの勝利！");
                  winner = 1;
                  return true;
            }
            /*else if (lion.rank() == 1) {  // ライオンがゴールした
                  System.out.println ("アンパンマンがゴール！");
                  System.out.println ("アンパンマンチームの勝利！");
                  winner = 1;
                  return true;
            } */
            else if (isMate (1)) { // E-ライオンが詰んだ
                  if (isChecked (1)) { // E-ライオンに王手がかかっている
                        System.out.println ("チェックメイト！");
                        System.out.println ("Aチームの勝利！");
                        winner = 1;
                  } else {
                        System.out.println ("スティールメイト！");
                        System.out.println ("引き分けです");
                        winner = 0;
                  }
                  return true;
            } else if (resign) {                // 投了した
                  System.out.println ("Bチームの勝利！");
                  winner = -1;
                  return true;
            } else if (checkPerpetual()) {
                  System.out.println ("千日手");
                  System.out.println ("引き分けです");
                  winner = 0;
                  return true;
            } else {
                  return false;
            }
      } else {                                 // Bチームの手番
            if (lion == null) {                // ライオンを取った
                  System.out.println ("Bチームの勝利！");
                  winner = -1;
```

```
                return true;
            } /*else if (e_lion.rank() == 6) { // E-ライオンがゴールした
                System.out.println ("E-ライオンがゴール！");
                System.out.println ("Bチームの勝利！");
                winner = -1;
                return true;
            } */
            else if (isMate (0)) { // Aチームが詰んだ
                if (isChecked (0)) { // Aに王手がかかっている
                    System.out.println ("チェックメイト！");
                    System.out.println ("Bチームの勝利！");
                    winner = -1;
                } else {
                    System.out.println ("スティールメイト！");
                    System.out.println ("引き分けです");
                    winner = 0;
                }
                return true;
            } else if (resign) {                // 投了した
                System.out.println ("Aチームの勝利！");
                winner = 1;
                return true;
            } else if (checkPerpetual()) {
                System.out.println ("千日手");
                System.out.println ("引き分けです");
                winner = 0;
                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 = lion.file(); // ライオンの座標
                  rank = lion.rank();

                  //自分の駒視点で左から時計回り
                  //E-犬_1 からの王手
                  if (board [rank][file-1] == E_DOG_1) return true;          //
左からの王手
                  else if (board [rank-1][file-1] == E_DOG_1) return
true;     // 左上からの王手
                  else if (board [rank-1][file] == E_DOG_1) return true;     //
上からの王手
                  else if (board [rank-1][file+1] == E_DOG_1) return true;    //
右上からの王手
                  else if (board [rank][file+1] == E_DOG_1) return true;      //
右からの王手
                  else if (board [rank+1][file] == E_DOG_1) return true;      //
下からの王手

                  //E-犬_2 からの王手
                  else if (board [rank][file-1] == E_DOG_2) return true;
// 左からの王手
                  else if (board [rank-1][file-1] == E_DOG_2) return
true;     // 左上からの王手
                  else if (board [rank-1][file] == E_DOG_2) return true;     //
上からの王手
                  else if (board [rank-1][file+1] == E_DOG_2) return true;    //
右上からの王手
                  else if (board [rank][file+1] == E_DOG_2) return true;      //
右からの王手
                  else if (board [rank+1][file] == E_DOG_2) return true;      //
下からの王手

                  //E-猫_1 からの王手
                  else if (board [rank-1][file-1] == E_NEKO_1) return
true;     // 左上からの王手
                  else if (board [rank-1][file] == E_NEKO_1) return true;     //
上からの王手
                  else if (board [rank-1][file+1] == E_NEKO_1) return
true;    //右上からの王手
```

```
                else if (board [rank+1][file+1] == E_NEKO_1) return true;   //
右下からの王手
                else if (board [rank+1][file-1] == E_NEKO_1) return true;   //
左下からの王手

                //E-猫_2 からの王手
                else if (board [rank-1][file-1] == E_NEKO_2) return
true;     // 左上からの王手
                else if (board [rank-1][file] == E_NEKO_2) return true;     //
上からの王手
                else if (board [rank-1][file+1] == E_NEKO_2) return
true;    //右上からの王手
                else if (board [rank+1][file+1] == E_NEKO_2) return true;   //
右下からの王手
                else if (board [rank+1][file-1] == E_NEKO_2) return true;   //
左下からの王手

                //E-ひよこ_1 からの王手
                else if (board [rank-1][file] == E_HIYOKO_1) return true;  //
上からの王手

                //E-ひよこ_2 からの王手
                else if (board [rank-1][file] == E_HIYOKO_2) return true;  //
上からの王手

                //E-ひよこ_3 からの王手
                else if (board [rank-1][file] == E_HIYOKO_3) return true;  //
上からの王手

                else return false;
            } else {
                file = e_lion.file(); // E-ライオンの座標
                rank = e_lion.rank();

                //自分の駒視点
                //犬_1 からの王手
                if (board [rank][file-1] == DOG_1) return true;          // 左
からの王手
                else if (board [rank-1][file] == DOG_1) return true;     // 上
からの王手
```

```
                else if (board [rank][file+1] == DOG_1) return true;      //右
からの王手
                else if (board [rank+1][file+1] == DOG_1) return true;    //右
下からの王手
                else if (board [rank+1][file] == DOG_1) return true;      //下
からの王手
                else if (board [rank+1][file-1] == DOG_1) return true;    //左
下からの王手

                //犬_2からの王手
                if (board [rank][file-1] == DOG_2) return true;          // 左
からの王手
                else if (board [rank-1][file] == DOG_2) return true;     // 上
からの王手
                else if (board [rank][file+1] == DOG_2) return true;      //右
からの王手
                else if (board [rank+1][file+1] == DOG_2) return true;    //右
下からの王手
                else if (board [rank+1][file] == DOG_2) return true;      //下
からの王手
                else if (board [rank+1][file-1] == DOG_2) return true;    //左
下からの王手

                //猫_1からの王手
                else if (board [rank-1][file-1] == NEKO_1) return true;     //
左上からの王手
                else if (board [rank-1][file+1] == NEKO_1) return true;    //
右上からの王手
                else if (board [rank+1][file+1] == NEKO_1) return true;   //右
下からの王手
                else if (board [rank+1][file] == NEKO_1) return true;       //
下からの王手
                else if (board [rank+1][file-1] == NEKO_1) return true;   //左
下からの王手

                //猫_2からの王手
                else if (board [rank-1][file-1] == NEKO_2) return true;     //
左上からの王手
                else if (board [rank-1][file+1] == NEKO_2) return true;    //
```

```
右上からの王手
                else if (board [rank+1][file+1] == NEKO_2) return true;   //右
下からの王手
                else if (board [rank+1][file] == NEKO_2) return true;       //
下からの王手
                else if (board [rank+1][file-1] == NEKO_2) return true;   //左
下からの王手

                //ひよこ_1 からの王手
                else if (board [rank+1][file] == HIYOKO_1) return true;  //下か
らの王手

                //ひよこ_2 からの王手
                else if (board [rank+1][file] == HIYOKO_2) return true;  //下か
らの王手

                //ひよこ_3 からの王手
                else if (board [rank+1][file] == HIYOKO_3) return true;  //下か
らの王手

                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) { // A チームの場合
                movableList = lion.movableList (board);                 // ライオ
ンが移動可能な手
                if (dog_1 != null) {                                    // 盤上にま
だ犬_1 がある場合
                    movableList.addAll (dog_1.movableList (board)); // 犬_1
が移動可能な手
                }
                if (dog_2 != null) {                                    // 盤上にま
だ犬_2 がある場合
                    movableList.addAll (dog_2.movableList (board)); // 犬_2
が移動可能な手
                }
                if (neko_1 != null) {                                    // 盤上に
まだ猫_1 がある場合
                    movableList.addAll (neko_1.movableList (board)); // 猫_1
が移動可能な手
                }
                if (neko_2 != null) {                                    // 盤上に
まだ猫_2 がある場合
                    movableList.addAll (neko_2.movableList (board)); // 猫_2
が移動可能な手
                }
                if (hiyoko_1 != null) {                                    // 盤上
にまだひよこ_1 がある場合
                    movableList.addAll (hiyoko_1.movableList (board)); // ひ
よこ_1 が移動可能な手
                }
                if (hiyoko_2 != null) {                                    // 盤上
にまだひよこ_2 がある場合
                    movableList.addAll (hiyoko_2.movableList (board)); // ひ
よこ_2 が移動可能な手
                }
                if (hiyoko_3 != null) {                                    // 盤上
にまだひよこ_3 がある場合
                    movableList.addAll (hiyoko_3.movableList (board)); // ひ
よこ_3 が移動可能な手
                }
                /*
```

```
                if (cat_1 != null) {                                    // 盤上にまだ成り猫_1 がある場合
                    movableList.addAll (cat_1.movableList (board)); // 成り猫_1 が移動可能な手
                }
                if (cat_2 != null) {                                    // 盤上にまだ成り猫_2 がある場合
                    movableList.addAll (cat_2.movableList (board)); // 成り猫_2 が移動可能な手
                }
                if (bird_1 != null) {                                    // 盤上にまだ成りひよこ_1 がある場合
                    movableList.addAll (bird_1.movableList (board)); // 成りひよこ_1 が移動可能な手
                }
                if (bird_2 != null) {                                    // 盤上にまだ成りひよこ_2 がある場合
                    movableList.addAll (bird_2.movableList (board)); // 成りひよこ_2 が移動可能な手
                }
                if (bird_3 != null) {                                    // 盤上にまだ成りひよこ_3 がある場合
                    movableList.addAll (bird_3.movableList (board)); // 成りひよこ_3 が移動可能な手
                }
                */

                if (isChecked(0)) { // ライオ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 {                // Bチームの場合
            movableList = e_lion.movableList (board);               // E-ライオンが移動可能な手
            if (e_dog_1 != null) {                                  // 盤上にまだE-犬_1がある場合
                movableList.addAll (e_dog_1.movableList (board)); // E-犬_1が移動可能な手
            }
            if (e_dog_2 != null) {                                  // 盤上にまだE-犬_2がある場合
                movableList.addAll (e_dog_2.movableList (board)); // E-犬_2が移動可能な手
            }
            if (e_neko_1 != null) {                                 // 盤上にまだE-猫_1がある場合
                movableList.addAll (e_neko_1.movableList (board)); // E-猫_1が移動可能な手
            }
            if (e_neko_2 != null) {                                 // 盤上にまだE-猫_2がある場合
                movableList.addAll (e_neko_2.movableList (board)); // E-猫_2が移動可能な手
            }
            if (e_hiyoko_1 != null) {                               // 盤上にまだE-ひよこ_1がある場合
                movableList.addAll (e_hiyoko_1.movableList (board)); // E-ひよこ_1が移動可能な手
            }
            if (e_hiyoko_2 != null) {                               // 盤上にまだE-ひよこ_2がある場合
                movableList.addAll (e_hiyoko_2.movableList (board)); // E-ひよこ_2が移動可能な手
            }
            if (e_hiyoko_3 != null) {                               // 盤上にまだE-ひよこ_3がある場合
                movableList.addAll (e_hiyoko_3.movableList (board)); // E-ひよこ_3が移動可能な手
```

```
                }
                /*
                if (e_cat_1 != null) {                                  // 盤上にまだE-成り猫_1がある場合
                        movableList.addAll (e_cat_1.movableList (board)); // E-成り猫_1が移動可能な手
                }
                if (e_cat_2 != null) {                                  // 盤上にまだE-成り猫_2がある場合
                        movableList.addAll (e_cat_2.movableList (board)); // E-成り猫_2が移動可能な手
                }
                if (e_bird_1 != null) {                                  // 盤上にまだE-成りひよこ_1がある場合
                        movableList.addAll (e_bird_1.movableList (board)); // E-成りひよこ_1が移動可能な手
                }
                if (e_bird_2 != null) {                                  // 盤上にまだE-成りひよこ_2がある場合
                        movableList.addAll (e_bird_2.movableList (board)); // E-成りひよこ_2が移動可能な手
                }
                if (e_bird_3 != null) {                                  // 盤上にまだE-成りひよこ_3がある場合
                        movableList.addAll (e_bird_3.movableList (board)); // E-成りひよこ_3が移動可能な手
                }
                */

                if (isChecked (1)) { // E-ライオ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;  //打つ手がないこと

        /* 現時点ですでに詰んでいるかどうかのチェック */
        if (playerNum == 0) { // A チームの手番
            if (lion == null) {              // ライオンが取られた
                checkmate = true;
                return Integer.MIN_VALUE; // 評価値無限小
            } else if (isMate (0)) { // A チームが詰んだ
                if (isChecked (0)) { // ライオ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) {                 // B チームが投了した
                return Integer.MAX_VALUE; // 評価値無限大
            }
        } else { // B チームの手番
            if (e_lion == null) {             // E-ライオンが取られた
                checkmate = true;
                return Integer.MAX_VALUE; // 評価値無限大
            } else if (isMate (1)) { // E-ライオンが詰んだ
                if (isChecked (1)) { // E-ライオ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) {                // A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 (lion.rank()) { // ライオン
case 1:
        value += 15;
        if(lion.file() == 3) ++value;
        break;
case 2:
        value += 13;
        if (lion.file() == 3) ++value;
        break;
case 3:
        value += 9;
        if (lion.file() == 3) ++value;
        break;
case 4:
        value += 7;
        if (lion.file() == 3) ++value;
        break;
case 5:
        value += 6;
        if (lion.file() == 3) ++value;
        break;
case 6 :
        value += 4;
        if(lion.file() == 3) ++value;
        break;
}

if (dog_1 != null) {        // 盤上に犬_1がある場合
        if (dog_1.rank() == 1)
            value += 4;
        else {
```

```
            value += 7;
            if (dog_1.file() == 3) ++value;
      }
}
if (dog_2 != null) {        // 盤上に犬_2 がある場合
      if (dog_2.rank() == 1)
            value += 4;
      else {
            value += 7;
            if (dog_2.file() == 3) ++value;
      }
}

if (neko_1 != null) {        // 盤上に猫_1 がある場合
      if (neko_1.rank() == 1)
            value += 2;
      else {
            value += 4;
            if (neko_1.file() == 3) ++value;
      }
}
if (neko_2 != null) {        // 盤上に猫_1 がある場合
      if (neko_2.rank() == 1)
            value += 2;
      else {
            value += 4;
            if (neko_2.file() == 3) ++value;
      }
}

if (hiyoko_1 != null) {        // 盤上にひよこ_1 がある場合
      if (hiyoko_1.rank() == 1) // ひよこは端まで進むと価値が下がる
            value += 0;
      else {
            value += 2;
            if (hiyoko_1.file() == 3) ++value;
      }
}
if (hiyoko_2 != null) {        // 盤上にひよこ_2 がある場合
      if (hiyoko_2.rank() == 1) // ひよこは端まで進むと価値が下がる
```

```
            value += 0;
      else {
            value += 2;
            if (hiyoko_2.file() == 3) ++value;
      }
}
if (hiyoko_3 != null) {       // 盤上にひよこ_3がある場合
      if (hiyoko_3.rank() == 1) // ひよこは端まで進むと価値が下がる
            value += 0;
      else {
            value += 2;
            if (hiyoko_3.file() == 3) ++value;
      }
}

switch (e_lion.rank()) { // E-ライオン
case 6 :
      value -= 15;
      if(e_lion.file() == 3) --value;
      break;
case 5:
      value -= 13;
      if(e_lion.file() == 3) --value;
      break;
case 4:
      value -= 9;
      if (e_lion.file() == 3) --value;
      break;
case 3:
      value -= 7;
      if (e_lion.file() == 3) --value;
      break;
case 2:
      value -= 6;
      if (e_lion.file() == 3) --value;
      break;
case 1:
      value -= 4;
      if (e_lion.file() == 3) --value;
      break;
}
```

```
if (e_dog_1 != null) {       // 盤上にE-犬_1がある場合
      if (e_dog_1.rank() == 6)
            value -= 4;
      else {
            value -= 7;
            if (e_dog_1.file() == 3) ++value;
      }
}
if (e_dog_2 != null) {       // 盤上にE-犬_2がある場合
      if (e_dog_2.rank() == 6)
            value -= 4;
      else {
            value -= 7;
            if (e_dog_2.file() == 3) ++value;
      }
}

if (e_neko_1 != null) {       // 盤上にE-猫_1がある場合
      if (e_neko_1.rank() == 6)
            value -= 2;
      else {
            value -= 4;
            if (e_neko_1.file() == 3) ++value;
      }
}
if (e_neko_2 != null) {       // 盤上にE-猫_2がある場合
      if (e_neko_2.rank() == 6)
            value -= 2;
      else {
            value -= 4;
            if (e_neko_2.file() == 3) ++value;
      }
}

if (hiyoko_1 != null) {       // 盤上にひよこ_1がある場合
      if (hiyoko_1.rank() == 6) // ひよこは端まで進むと価値が下がる
            value -= 0;
      else {
            value -= 2;
            if (hiyoko_1.file() == 3) ++value;
```

```
            }
        }
        if (hiyoko_2 != null) {        // 盤上にひよこ_2がある場合
            if (hiyoko_2.rank() == 6) // ひよこは端まで進むと価値が下がる
                value -= 0;
            else {
                value -= 2;
                if (hiyoko_2.file() == 3) ++value;
            }
        }
        if (hiyoko_3 != null) {        // 盤上にひよこ_3がある場合
            if (hiyoko_3.rank() == 6) // ひよこは端まで進むと価値が下がる
                value -= 0;
            else {
                value -= 2;
                if (hiyoko_3.file() == 3) ++value;
            }
        }

        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) {  // 一定手数まで読んでいる場合はそれ以上先読みしない
            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
```
            if (precheckPerpetual()) {
                  // System.out.println (moves + "+" + lookAheadMoves + ":" +
"parpetual");
                  return 0; // 先読み開始後同一局面になる場合は評価を停止する
            }
            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

            ++lookAheadMoves; // 先読みのために手数を 1 増やす
            if (playerNum == 0) { // A 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
                              --lookAheadMoves; // 先読みが終了したので手数を 1 戻す
                              return Integer.MAX_VALUE;
                        }
                  }
            } else { // B 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
                              --lookAheadMoves; // 先読みが終了したので手数を 1 戻す
                              return Integer.MIN_VALUE;
                        }
                  }
            }

            --lookAheadMoves; // 先読みが終了したので手数を 1 戻す
            Random rnd = new Random();
            int ran = rnd.nextInt (3); // 0～2 の乱数を生成
            if (bestValue != Integer.MAX_VALUE && bestValue !=
Integer.MIN_VALUE) {
                  bestValue = bestValue + ran - 1; // 評価値に乱数を加える
            }
            //System.out.println (moves + "+" + lookAheadMoves + ":" + bestMove
+ "[" + bestValue + "]");
            return bestValue;
      }

      /**
       * 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 LION :
      movingPiece = nextBoard.lion;
      break;
case DOG_1 :
      movingPiece = nextBoard.dog_1;
      break;
case DOG_2 :
      movingPiece = nextBoard.dog_2;
      break;
case NEKO_1 :
      movingPiece = nextBoard.neko_1;
      break;
case NEKO_2 :
      movingPiece = nextBoard.neko_2;
      break;
case HIYOKO_1 :
      movingPiece = nextBoard.hiyoko_1;
      break;
case HIYOKO_2 :
      movingPiece = nextBoard.hiyoko_2;
      break;
case HIYOKO_3 :
      movingPiece = nextBoard.hiyoko_3;
      break;
      /*
case CAT_1 :
      movingPiece = nextBoard.cat_1;
      break;
case CAT_2 :
      movingPiece = nextBoard.cat_2;
      break;
case BIRD_1 :
      movingPiece = nextBoard.bird_1;
      break;
case BIRD_2 :
      movingPiece = nextBoard.bird_2;
      break;
case BIRD_3 :
```

```
        movingPiece = nextBoard.bird_3;
        break;
        */

case E_LION :
        movingPiece = nextBoard.e_lion;
        break;
case E_DOG_1 :
        movingPiece = nextBoard.e_dog_1;
        break;
case E_DOG_2 :
        movingPiece = nextBoard.e_dog_2;
        break;
case E_NEKO_1 :
        movingPiece = nextBoard.e_neko_1;
        break;
case E_NEKO_2 :
        movingPiece = nextBoard.e_neko_2;
        break;
case E_HIYOKO_1 :
        movingPiece = nextBoard.e_hiyoko_1;
        break;
case E_HIYOKO_2 :
        movingPiece = nextBoard.e_hiyoko_2;
        break;
case E_HIYOKO_3 :
        movingPiece = nextBoard.e_hiyoko_3;
        break;
        /*
case E_CAT_1 :
        movingPiece = nextBoard.e_cat_1;
        break;
case E_CAT_2 :
        movingPiece = nextBoard.e_cat_2;
        break;
case E_BIRD_1 :
        movingPiece = nextBoard.e_bird_1;
        break;
case E_BIRD_2 :
        movingPiece = nextBoard.e_bird_2;
        break;
```

```
		case E_BIRD_3 :
			movingPiece = nextBoard.e_bird_3;
			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 LION :
			return "ライオン";
		case DOG_1 :
			return "犬_1";
		case DOG_2 :
			return "犬_2";
		case NEKO_1 :
			return "猫_1";
		case NEKO_2 :
			return "猫_2";
		case HIYOKO_1 :
			return "ひよこ_1";
		case HIYOKO_2 :
			return "ひよこ_2";
		case HIYOKO_3 :
			return "ひよこ_3";
			/*
		case CAT_1 :
```

```
        return "成り猫_1";
    case CAT_2 :
        return "成り猫_2";
    case BIRD_1 :
        return "成りひよこ_1";
    case BIRD_2 :
        return "成りひよこ_2";
    case BIRD_3 :
        return "成りひよこ_3";
        */

    case E_LION :
        return "ライオン";
    case E_DOG_1 :
        return "犬_1";
    case E_DOG_2 :
        return "犬_2";
    case E_NEKO_1 :
        return "猫_1";
    case E_NEKO_2 :
        return "猫_2";
    case E_HIYOKO_1 :
        return "ひよこ_1";
    case E_HIYOKO_2 :
        return "ひよこ_2";
    case E_HIYOKO_3 :
        return "ひよこ_3";
        /*
    case E_CAT_1 :
        return "成り猫_1";
    case E_CAT_2 :
        return "成り猫_2";
    case E_BIRD_1 :
        return "成りひよこ_1";
    case E_BIRD_2 :
        return "成りひよこ_2";
    case E_BIRD_3 :
        return "成りひよこ_3";
        */
    default :
```

```
                return "?";
            }
        }

        /**
         * 現在の盤面を記憶
         * 現在の盤面を配列boardRecordに記憶する
         */
        void recordBoard() {
            if (moves + lookAheadMoves < maxMove) {
                for (int r=0; r<6; ++r) {
                    for (int f=0; f<5; ++f) {
                        boardRecord [moves + lookAheadMoves][r][f] =
board[r+1][f+1];
                    }
                }
            }
        }

        /**
         * 千日手になったか判定
         * 同じ局面が3回出てくれば千日手と見做す
         * @return 千日手か
         */
        boolean checkPerpetual () {
            int counter = 0;
            for (int i=moves-3; i>=0; i-=2) {
                int match = 0;
                for (int r=0; r<6; ++r) {
                    for (int f=0; f<5; ++f) {
                        if (boardRecord [i][r][f] == board[r+1][f+1]) {
                            ++match;
                        }
                    }
                }
                if (match == 30) {
                    ++counter;
                    if (counter >= 3) return true;
                }
            }
            return false;
```

```
	}

	/**
	 * 先読み時に同じ局面になったか判定
	 * 先読み開始以降同じ局面が出てくればそれ以上先読みはしない
	 * @return 同一局面か
	 */
	boolean precheckPerpetual () {
		int m = moves-6;
		if (m<0) m=0;
		for (int i=moves+lookAheadMoves-1; i>=m; --i) {
			int match = 0;
			for (int r=0; r<6; ++r) {
				for (int f=0; f<5; ++f) {
					if (boardRecord [i][r][f] == board[r+1][f+1]) {
						++match;
					}
				}
			}
			if (match == 30) {
				// System.out.print (moves + "+" + lookAheadMoves + ":"
+ i + "parpetual");
				return true;
			}
		}
		return false;
	}

	/**
	 * 先読みの上限数を指定
	 * @param d 先読みの上限数
	 */
	public void setMaxDepth (int d) {
		maxDepth = d;
	}

	/**
	 * 手数をリセット
	 */
	public void resetMoves () {
		moves = 0;
```

```
            lookAheadMoves = 0;
        }

        /**
         * 勝者の番号を帰す
         * 1:先手勝ち -1:後手勝ち 0:引き分け
         * @return 勝者の番号
         */
        public int winner() {
            return winner;
        }
}
```

クラス NextMove

```
package gorogoro;

/**
 * 駒の移動可能な位置を表すクラス
 */

public class NextMove {
        //A チーム　先手の駒　盤上手前
        final static int LION = 1; //ライオン
        final static int DOG_1 = 2; //犬_1
        final static int DOG_2 = 3; //犬_2
        final static int NEKO_1 = 4; //猫_1
        final static int NEKO_2 = 5; //猫_2
        final static int HIYOKO_1 = 6; //ひよこ_1
        final static int HIYOKO_2 = 7; //ひよこ_2
        final static int HIYOKO_3 = 8; //ひよこ_3
        /*
        final static int CAT_1 = 9; //成り猫_1
        final static int CAT_2 = 10; //成り猫_2
        final static int BIRD_1 = 11; //成りひよこ_1
        final static int BIRD_2 = 12; //成りひよこ_2
        final static int BIRD_3 = 13; //成りひよこ_3
        */

        //B チーム　後手の駒　盤上奥
```

```
	final static int E_LION = -1; //ライオン
	final static int E_DOG_1 = -2; //犬_1
	final static int E_DOG_2 = -3; //犬_2
	final static int E_NEKO_1 = -4; //猫_1
	final static int E_NEKO_2 = -5; //猫_2
	final static int E_HIYOKO_1 = -6; //ひよこ_1
	final static int E_HIYOKO_2 = -7; //ひよこ_2
	final static int E_HIYOKO_3 = -8; //ひよこ_3
	/*
	final static int E_CAT_1 = -9; //成り猫_1
	final static int E_CAT_2 = -10; //成り猫_2
	final static int E_BIRD_1 = -11; //成りひよこ_1
	final static int E_BIRD_2 = -12; //成りひよこ_12
	final static int E_BIRD_3 = -13; //成りひよこ_13
	*/


	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;
}

/**
 * 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 LION : score = "L"; break;
case DOG_1 : score = "D1"; break;
case DOG_2 : score = "D2"; break;
case NEKO_1 : score = "N1"; break;
case NEKO_2 : score = "N2"; break;
case HIYOKO_1 : score = "H1"; break;
case HIYOKO_2 : score = "H2"; break;
case HIYOKO_3 : score = "H3"; break;
/*
case CAT_1 : score = "C1"; break;
case CAT_2 : score = "C2"; break;
case BIRD_1 : score = "B1"; break;
case BIRD_2 : score = "B2"; break;
case BIRD_3 : score = "B3"; break;
*/

case E_LION : score ="EL"; break;
case E_DOG_1 : score = "ED1"; break;
case E_DOG_2 : score = "ED2"; break;
case E_NEKO_1 : score = "EN1"; break;
case E_NEKO_2 : score = "EN2"; break;
case E_HIYOKO_1 : score = "EH1"; break;
case E_HIYOKO_2 : score ="EH2"; break;
case E_HIYOKO_3 : score = "EH3"; break;
/*
case E_CAT_1 : score = "EC1"; break;
case E_CAT_2 : score = "EC2"; break;
case E_BIRD_1 : score = "EB1"; break;
case E_BIRD_2 : score = "EB2"; break;
case E_BIRD_3 : score = "EB3"; break;
*/
default : score = "?"; break;
}

switch(nextFile) {
case 1 : score += "a"; break;
case 2 : score += "b"; break;
case 3 : score += "c"; break;
case 4 : score += "d"; break;
case 5 : score += "e"; 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;
        case 6 : score += "6"; break;
        default : score += "?"; break;
        }
}
//将棋風の棋譜を作成
else {
        switch(nextFile) {
        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;
        }
        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;
        case 6 : score += "六"; break;
        default : score += "？" ; break;
        }
        switch(type) {
        case LION       : score += "ら　"; break; //ライオン
        case DOG_1      : score += "い_1 "; break; //犬_1
        case DOG_2      : score += "い_2 "; break; //犬_2
        case NEKO_1     : score += "ね_1 "; break; //猫_1
        case NEKO_2     : score += "ね_2 "; break; //猫_2
        case HIYOKO_1   : score += "ひ_1 "; break; //ひよこ_1
        case HIYOKO_2   : score += "ひ_2 "; break; //ひよこ_2
```

```
                case HIYOKO_3   : score += "ひ_3  "; break; //ひよこ_3
                /*
                case CAT_1      : score += "き_1  "; break; //成り猫_1
                case CAT_2      : score += "き_2  "; break; //成り猫_2
                case BIRD_1     : score += "と_1  "; break; //成りひよこ_1
                case BIRD_2     : score += "と_2  "; break; //成りひよこ_2
                case BIRD_3     : score += "と_3  "; break; //成りひよこ_3
                */

                case E_LION     : score += "ラ    "; break; //ライオン
                case E_DOG_1    : score += "イ_1  "; break; //犬_1
                case E_DOG_2    : score += "イ_2  "; break; //犬_2
                case E_NEKO_1   : score += "ネ_1  "; break; //猫_1
                case E_NEKO_2   : score += "ネ_2  "; break; //猫_2
                case E_HIYOKO_1 : score += "ヒ_1  "; break; //ひよこ_1
                case E_HIYOKO_2 : score += "ヒ_2  "; break; //ひよこ_2
                case E_HIYOKO_3 : score += "ヒ_3  "; break; //ひよこ_3
                /*
                case E_CAT_1    : score += "キ_1  "; break; //成り猫_1
                case E_CAT_2    : score += "キ_2  "; break; //成り猫2
                case E_BIRD_1   : score += "ト_1  "; break; //成りひよこ_1
                case E_BIRD_2   : score += "ト_2  "; break; //成りひよこ_2
                case E_BIRD_3   : score += "ト_3  "; break; //成りひよこ_3
                */
                default : score += "?"; break;
                }
            }
            return score;
        }
}
```

クラス Piece

```
package gorogoro;

/**
 * 駒の移動可能な位置を表すクラス
 */

public class NextMove {
```

```
	//A チーム　先手の駒　盤上手前
	final static int LION = 1; //ライオン
	final static int DOG_1 = 2; //犬_1
	final static int DOG_2 = 3; //犬_2
	final static int NEKO_1 = 4; //猫_1
	final static int NEKO_2 = 5; //猫_2
	final static int HIYOKO_1 = 6; //ひよこ_1
	final static int HIYOKO_2 = 7; //ひよこ_2
	final static int HIYOKO_3 = 8; //ひよこ_3
	/*
	final static int CAT_1 = 9; //成り猫_1
	final static int CAT_2 = 10; //成り猫_2
	final static int BIRD_1 = 11; //成りひよこ_1
	final static int BIRD_2 = 12; //成りひよこ_2
	final static int BIRD_3 = 13; //成りひよこ_3
	*/

	//B チーム　後手の駒　盤上奥
	final static int E_LION = -1; //ライオン
	final static int E_DOG_1 = -2; //犬_1
	final static int E_DOG_2 = -3; //犬_2
	final static int E_NEKO_1 = -4; //猫_1
	final static int E_NEKO_2 = -5; //猫_2
	final static int E_HIYOKO_1 = -6; //ひよこ_1
	final static int E_HIYOKO_2 = -7; //ひよこ_2
	final static int E_HIYOKO_3 = -8; //ひよこ_3
	/*
	final static int E_CAT_1 = -9; //成り猫_1
	final static int E_CAT_2 = -10; //成り猫_2
	final static int E_BIRD_1 = -11; //成りひよこ_1
	final static int E_BIRD_2 = -12; //成りひよこ_12
	final static int E_BIRD_3 = -13; //成りひよこ_13
	*/


	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;
}

/**
 * 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 LION : score = "L"; break;
		case DOG_1 : score = "D1"; break;
		case DOG_2 : score = "D2"; break;
		case NEKO_1 : score = "N1"; break;
		case NEKO_2 : score = "N2"; break;
		case HIYOKO_1 : score = "H1"; break;
		case HIYOKO_2 : score = "H2"; break;
		case HIYOKO_3 : score = "H3"; break;
		/*
		case CAT_1 : score = "C1"; break;
		case CAT_2 : score = "C2"; break;
		case BIRD_1 : score = "B1"; break;
		case BIRD_2 : score = "B2"; break;
		case BIRD_3 : score = "B3"; break;
		*/

		case E_LION : score ="EL"; break;
		case E_DOG_1 : score = "ED1"; break;
		case E_DOG_2 : score = "ED2"; break;
		case E_NEKO_1 : score = "EN1"; break;
		case E_NEKO_2 : score = "EN2"; break;
```

```
        case E_HIYOKO_1 : score = "EH1"; break;
        case E_HIYOKO_2 : score ="EH2"; break;
        case E_HIYOKO_3 : score = "EH3"; break;
        /*
        case E_CAT_1 : score = "EC1"; break;
        case E_CAT_2 : score = "EC2"; break;
        case E_BIRD_1 : score = "EB1"; break;
        case E_BIRD_2 : score = "EB2"; break;
        case E_BIRD_3 : score = "EB3"; break;
        */
        default : score = "?"; break;
        }

        switch(nextFile) {
        case 1 : score += "a"; break;
        case 2 : score += "b"; break;
        case 3 : score += "c"; break;
        case 4 : score += "d"; break;
        case 5 : score += "e"; 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;
        case 6 : score += "6"; break;
        default : score += "?"; break;
        }
}
//将棋風の棋譜を作成
else {
        switch(nextFile) {
        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;
```

```
}
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;
case 6 : score += "六"; break;
default : score += "？" ; break;
}
switch(type) {
case LION       : score += "ら   "; break; //ライオン
case DOG_1      : score += "い_1  "; break; //犬_1
case DOG_2      : score += "い_2  "; break; //犬_2
case NEKO_1     : score += "ね_1  "; break; //猫_1
case NEKO_2     : score += "ね_2  "; break; //猫_2
case HIYOKO_1   : score += "ひ_1  "; break; //ひよこ_1
case HIYOKO_2   : score += "ひ_2  "; break; //ひよこ_2
case HIYOKO_3   : score += "ひ_3  "; break; //ひよこ_3
/*
case CAT_1      : score += "き_1  "; break; //成り猫_1
case CAT_2      : score += "き_2  "; break; //成り猫_2
case BIRD_1     : score += "と_1  "; break; //成りひよこ_1
case BIRD_2     : score += "と_2  "; break; //成りひよこ_2
case BIRD_3     : score += "と_3  "; break; //成りひよこ_3
*/

case E_LION     : score += "ラ   "; break; //ライオン
case E_DOG_1    : score += "イ_1  "; break; //犬_1
case E_DOG_2    : score += "イ_2  "; break; //犬_2
case E_NEKO_1   : score += "ネ_1  "; break; //猫_1
case E_NEKO_2   : score += "ネ_2  "; break; //猫_2
case E_HIYOKO_1 : score += "ヒ_1  "; break; //ひよこ_1
case E_HIYOKO_2 : score += "ヒ_2  "; break; //ひよこ_2
case E_HIYOKO_3 : score += "ヒ_3  "; break; //ひよこ_3
/*
case E_CAT_1    : score += "キ_1  "; break; //成り猫_1
case E_CAT_2    : score += "キ_2  "; break; //成り猫2
case E_BIRD_1   : score += "ト_1  "; break; //成りひよこ_1
```

```
                case E_BIRD_2   : score += "ト_2  "; break; //成りひよこ_2
                case E_BIRD_3   : score += "ト_3  "; break; //成りひよこ_3
                */
                default : score += "?"; break;
                }
            }
            return score;
        }
}
```