卒業研究報告書

題目

ごろごろ動物将棋の解析

指導教員

石水　隆　講師

報告者

18–1–037–0131

玉井　睦也

近畿大学理工学部情報学科

令和 04 年 1 月 23 日提出

概要

「ごろごろどうぶつしょうぎ」(以下ごろごろ動物将棋とする) は女流棋士の北尾まどかがルールを考案したボードゲームである．ごろごろ動物将棋は 5x6 のゲーム盤と, 本将棋の王将, 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．盤面が小さいこと，使用する駒の種類が少ないことを除けばごろごろ動物将棋のルールは本将棋と同じであり，王将に相当する駒であるライオンを詰めることが勝利条件である．

ごろごろ動物将棋の前作にあたるどうぶつしょうぎ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示されている．しかし，ごろごろ動物将棋はまだ完全解析はされていない．そこで本研究では, ごろごろ動物将棋の完全解析を最終目標とし, ごろごろ将棋のゲーム AI プログラムを Java を用いて作成する．ただし, 本研究で作成するゲーム AI プログラムはゲームの性質上盤が小さく駒が取られやすいことで持ち駒の出し合うことが多く発生し千日手が発生する原因になったため，持ち駒なしとして作成した．

本研究ので作成するゲーム AI は, ある局面の全ての合法手を対象に数手先まで先読みを行い, その結果をもとに各手に評価値を求め, 評価値が最大である手を指す．評価値は先読みした局面での盤上の駒の数と着手可能手数から算出する．

本研究で作成した AI 同士の対戦を 300 回行ったところ, 先手勝ち 132 回, 後手勝ち 90 回, 引き分け 78 回という結果を得られた．持ち駒がないことからゲームの後半着手可能手が限られることが多発し千日手となる場合が多くなるという結果となった．しかし, 上記の結果から先手有利だと推測される．

目次

1　序論

1.1　ごろごろどうぶつしょうぎとは

ごろごろどうぶつしょうぎ (以下ごろごろ動物将棋とする) は 2012 年 11 月 15 日に発売されたミニ将棋の一種である [5].

ごろごろ動物将棋は 5x6 のゲーム盤と, 本将棋の王将, 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．盤面が小さいこと，使用する駒の種類が少ないことを除けばごろごろ動物将棋のルールは本将棋と同じであり，王将に相当する駒であるライオンを詰めることが勝利条件である．

ごろごろ動物将棋は前作のどうぶつしょうぎのルールを拡張したものであり, ルールは女流棋士の北尾まどかや藤田麻衣子によって考案された．将棋をほとんど知らない子供達向けに作られてたミニ将棋である．本将棋と異なる点は本将棋が 9x9 の盤を使用するのに対し, ごろごろ動物将棋は 5x6 の盤を使用しまた．飛角などのいわゆる走り駒が無く，どの駒も可動範囲が最大 1 マスとなっている．

1.2　ミニ将棋に関する既知の結果

ごろごろ動物将棋に類似したミニ将棋としてどうぶつしょうぎ (以下動物将棋とする)，アンパンマンはじめてしょうぎ (以下アンパンマン将棋とする) などがある [1, 2].

動物将棋はごろごろ動物将棋の前作にあたる子供向けのミニ将棋である．図 1 に動物将棋の初期配置を示す．動物将棋は，3x4 のゲーム盤と，ライオン，キリン，象，ひよこの 4 つの駒を用いる．また，本将棋と同様に取った相手の駒は，持駒として打つことができる．動物将棋は田中により完全解析されており，双方最善手を指すと 78 手で後手が勝つことが示されている [1].

アンパンマン将棋は，3x5 のゲーム盤と，アンパンマン，食パンマン，カレーパンマン，ばいきんまん，ホラーマン，ドキンちゃんの 6 種類の駒を用いるミニ将棋である．図 2 にアンパンマン将棋の初期配置を示す．本将棋や動物将棋と異なり，アンパンマン将棋は取った相手の駒は打つことができず取り捨てとなる．アンパンマン将棋は，塩田らにより双方最善手を指すと引き分けとなることが示されている [2].

図 1　動物将棋の初期配置

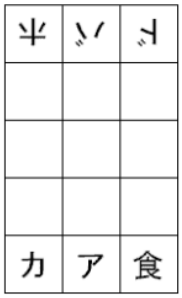

図 2　アンパンマン将棋の初期配置

1.3　ごろごろどうぶつ将棋に関する既知の結果

ごろごろ動物将棋はみんなのどうぶつしょうぎの一部コンテンツとして 2019 年に (C)SilverStarJapan から Playstation4 と Nintendo switch を対象に発売された [3, 4]. COM の難易度は 8 段階あり, 着手可能マスは色が変わっており全体的に分かりやすいデザインとなっている. ルールに関しては動画が用意されており理解しやすいものとなっており, ユーザインタフェースでは大半が平仮名とカタカナを使用しており子供向けに作成されている.

1.4　本研究の目的

1.2 節で述べた通り，前作である動物将棋は完全解析されており最善手を指すことができ，その場合後手が勝つことが分かっている．一方，ごろごろ動物将棋は動物将棋よりもゲーム盤が広く使用する駒の数が多いため現在も完全解析されておらず，先手後手のどちらが有利であるかは分かっていない。そこで本研究ではごろごろ動物将棋が先手有利か後手有利かを検証するために, ごろごろ動物将棋のゲーム AI プログラムを作成し, AI 同士で対戦させ先手と後手の勝数を求める.

1.5　本報告書の構成

本報告書の構成は以下の通りである. 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. 第 3 章では本研究で作成したプログラムの説明や評価値の計算方法について詳しく説明をする. そして第 4 章では研究結果を述べ, 第 5 章では結論と今後の課題について述べる.

2　ごろごろ動物将棋

本章ではごろごろ動物将棋について説明する.

2.1　将棋盤と駒

ごろごろ動物将棋は横 5 マス縦 6 マスの計 30 マスの将棋盤を使用し, 駒は王将, 金将, 銀将, 歩兵に相当するライオン, 犬, 猫, ひよこの 4 種類の駒で遊ぶミニ将棋の一種である. 図 3 にごろごろ動物将棋の初期配置を示す.

2.2　ゲーム進行と勝敗条件

ごろごろ動物将棋はミニ将棋の一種である 2 人用ボードゲームであり，基本的なルールは本将棋と同様である. 双方のプレイヤーに交互に手番がまわり，手番プレイヤーは盤上の自駒を動かすか，持駒を打つことができる．盤面手前が先手, 奥が後手とする.

ごろごろ将棋は，ゲーム盤の一，二段目が先手の陣地，五，六段目が後手の陣地となり，本将棋と同様に，敵陣に移動した駒，または敵陣から移動した駒は成ることができる．成れる駒は猫, ひよこの 2 種類の駒であり, 駒が成った場合, ひよこはと金に相当する鶏, 猫は成銀に相当するパワーアップ猫となる.

図 3　ごろごろ動物将棋の初期配置

本将棋と同様に, 敵駒を取った場合は持駒にすることができ, 自分の手番で好きな場所に打つことができる. ただし持駒を打つ場合, 犬と猫はどの空きマスでも打てるが, ひよこは先手の場合ゲーム盤の六段目, 後手の場合は一段目に打つことができない. また, 本将棋の二歩の禁止および打ち歩詰めの禁止と同様に，ひよこを打つ場合はすでに自駒のひよこがある列には打つことはできず, また, 持駒のひよこを打って詰みにすることは反則手とされている.

ごろごろ動物将棋は相手のライオンを詰ますことが勝利条件であるまた，対戦中同一局面が 3 回起こった場合は千日手とし引き分けとなる.

2.3　ごろごろどうぶつしょうぎの総局面数

ごろごろ動物将棋の総局面数について考える. 先手のライオンは常に盤上にあるため 30 通り, 後手のライオンは先手のライオンと隣接することはないため後手のライオンを置けるマスの数は最大で 26 通りとなり, 合わせて 30x26 通りとなる. 次に犬は先手後手とも 2 つずつ持っており, 先手側は 28 通り, 後手側は 28 通りとなり, 先手と後手の持ち駒が各 1 通りなので計 58 通りとなり, 犬は 4 つ存在するので 58^4 通りとなる. ただし 4 つの駒に区別がないので $58^4/24$ 通りとなる. 猫も同様に求めると $58^4/24$ 通りとなり, ひよこも同様に求めると $58^6/720$ 通りとなる. 以上のことから, ごろごろ動物将棋の総局面数は約 $1.0\text{x}10^{22}$ 通りである. 動物将棋の総局面数は 1,567,925,964 通り [1] と比較すると, 約 $6.0\text{x}10^{12}$ 倍局面数が多いことが分かる.

3　研究内容

本研究では, ごろごろ動物将棋の完全解析のため Java を用いた AI 同士の対戦が可能なプログラムを作成する. ただし, ただし, 本研究で作成するゲーム AI プログラムはゲームの性質上盤が小さく駒が取られやすいこ

とで持ち駒の出し合うことが多く発生し千日手が発生する原因になったため，持ち駒なしとし，取った相手の駒は取り捨てとする．

本研究で作成する AI は着手可能手から得られる局面を先読みし, 算出された評価値を各手の評価値とし, 最も評価の高い手を指すようにしている．

3.1 着手可能の判断方法

ある局面での着手可能手は, 自駒が移動可能かどうかの判定を行うことで求められる．ただし, 自駒があるマスや盤外には移動できないものとする．また, ライオンは自殺手を防ぐために相手駒に取られるマスへ移動不可能とする．そして王手がかけられた場合は, 王手を回避することが出来る手のみを着手可能手としそれ以外の手は除外するものとする．最後に着手可能手が存在しない場合は詰みと判断し負けとなる．

3.2 評価値の計算

将棋は自駒が多いほど有利だとされているので, 評価値は盤上にある駒の種類と位置により決定される．ただし, ライオンは位置によって評価は変わらないものとする．また, 本将棋では攻めを銀将, 守りを金将とする考えがあるため銀将である猫は敵ライオンとの距離によって評価値が加算され, 金将である犬は自ライオンとの距離によって評価値が加算されるようになっており, 鶏とパワーアップ猫は基本的に敵陣近くにあり攻めの駒として考えるため猫同様に敵ライオンとの距離によって評価値が加算される．先手側の駒を正の値, 後手側の駒を負の値として駒の価値を設定し, 駒の価値の合計値を駒の種類による評価値として用いる．また，一般的に着手可能手数が多いほど有利とされているため着手可能手数による評価値は数に応じて評価を高くするものとする．また, 着手可能手の中に勝ちを確立する手がある場合はその局面の評価値を無限大, 負けの場合は無限小とする．先読み時に同じ局面が 3 回出てきた場合は千日手と判断し評価値を 0 とする．表 1 に各駒の評価値, 表 2 に犬と自ライオンとの位置関係による評価値, 表 3 に猫, 鶏, パワーアップ猫と敵ライオンとの位置関係による評価値, 表 4 に着手可能手数の評価値を示す．また，王手を掛けた場合，相手の手を王手から逃れる手のみに制限できるので王手を掛けるのは有利とされている．そこで王手を掛けた場合は評価値に有利となるように修正を加える．

以上の評価基準により，ある局面の評価値の計算方法は

$$\text{局面の評価値} = (\text{各駒の評価値}) + (\text{各駒の位置}) + (\text{着手可能手数}) + (\text{王手}\ (\text{先手} + 6, \text{後手} - 6))$$

となっている．

表 1　各駒の評価値

駒	ひよこ	猫	犬	鶏	パワーアップ猫	ライオン
評価値	1	4	4	4	4	15

表 2　犬:位置による評価値

自ライオンとの距離	1	2	3 以上
評価値	2	1	0

表 3　猫, 鶏, パワーアップ猫:位置による評価値

敵ライオンとの距離	1	2	3 以上
評価値	2	1	0

表 4　着手可能手数の評価値

着手可能手	0	1	2	3	・・・
評価値	0	1	2	3	・・・

4　ごろごろ動物将棋プログラム

本章では, ごろごろ動物将棋プログラムについて述べる. 付録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.

4.1　プログラムの仕様

本研究で作成したごろごろ動物将棋のプログラムは AI 対 AI 戦のみでプレイヤーが駒を動かすことはできない. また, 先読み数も調節可能である. 調整する場合は Board クラスのフィールドにある maxDepth の値を変えることで先読み数を変更することができる. 待ったの機能はなく, 着手可能手がない場合は詰みと判定され投了する処理が行われる.

ゲームの起動方法は以下の手順で起動することができる.

1. 付録にあるソースコードをコピーする.
2. それぞれのファイル名は GorogoroSyogi.java, Board.java, NextMove.java, Piece.java と保存する.
3. Windows 環境ならコマンドプロンプト, Mac 環境ならターミナルを起動する.
4.「cd ファイルの保存場所」と打ち Enter キーを押す.(ファイルの保存場所は Downloads や Desktop などファイルがあるフォルダ名を打つ)
5.「java GorogoroSyogi」と打ちゲームを起動する.

メインクラスである GorogoroSyogi クラスを実行することでゲームはスタートされる. 1 順目の流れとしては次のようになっている.

1. 先読み数を入力する
2. 盤面の表示 (Board クラスの showBoard メソッド)
3. 先手の着手可能手リストの作成 (Board クラスの createMovableList メソッド)
4. 王手の状況であるかを調べる (Board クラスの isChecked メソッド)
5. 詰みの状況であるかを調べる (Board クラスの checkWin メソッド)
6. 先読みを行い, 最も高評価な手を選び駒を移動させ, 棋譜を表示する.(Board クラスの com メソッド)
7. 盤面の表示
8. 後手の着手可能手リストの作成
9. 王手の状況であるかを調べる

10. 詰みの状況であるかを調べる
11. 先読みを行い, 最も低評価な手を選び駒を移動させ, 棋譜を表示する.

また, isChecked メソッドで王手の場合は「王手!」と表示され, checkWin メソッドで詰みと判定された場合は「先手の勝利!」や「後手の勝利!」, 千日手の場合は「引き分けです」と表示される.

4.2 GorogoroSyogi クラス

GorogoroSyogi クラスはゲームを進行するメインクラスである.

4.3 Board クラス

Board クラスはゲームの盤面を管理するクラスである. 表 5 に Board クラスの各メソッドについてまとめる.

表 5 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, 駒のインスタンス化
Board(int[ ][ ])	コンストラクタ, 盤面の更新
void showBoard()	盤面を表示する
void showPiece()	駒の種類を表示する
String com(int)	最良の手を指し棋譜を返す
String movePiece(Piece, int, int, int)	駒を移動させ棋譜を返す
void removePiece(int, int)	特定の座標の駒を削除する
void promotePiece(int[ ][ ])	駒を成る
boolean checkWin(int)	勝敗がついたかの判定
boolean isMate(int)	詰みかどうかの判定
boolean isChecked(int)	王手かどうかの判定
void createMovableList(int)	着手可能手を作成する
int value(int)	盤面の評価値を計算し返す
int value(int, int)	一定数先読みを行い評価値を返す
Board nextBoard(NextMove, int)	駒を移動させた後の盤面を返す
String name(int)	駒の種類を名前で返す
void recordBoard()	現在の盤面を記録する
boolean checkPerpetual()	千日手かどうかの判定
void resetMoves()	手数をリセットする
int winner()	商社の番号を返す

4.4 NextMove クラス

NextMove クラスは駒が移動したときに，その駒の情報を更新するクラスである．移動した駒の種類, 移動先の座標, 評価値を更新または返すメソッドがある．表 6 に NextMove クラスの各メソッドについてまとめる．

表 6　NextMove クラスのメソッド

NextMove(int, int, int)	移動する駒と移動先の座標を更新する
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

4.5 Piece クラス

Piece クラスは駒を管理するクラスである．各駒の移動可能方向や初期位置, 特定の座標への移動, 移動可能なマスを特定しリストとして返すメソッドがある．表 7 には Piece クラスの各メソッドについてまとめる．

表 7　Piece クラスのメソッド

Piece(int)	コンストラクタ, 駒の初期設定を行う
Piece(int, int, int)	コンストラクタ, 特定の座標に駒を登録する
void setMovableVector()	駒の移動可能方向を設定する
void setInitialPosition()	駒の初期位置を設定する
void setPosition(int, int)	駒を特定の座標に設定する
void removeFromBoard()	駒を盤上から削除する
boolean isOnBoard()	駒が盤上にあるかの判定
int file()	駒の x 座標を返す
int rank()	駒の y 座標を返す
int type()	駒の種類を返す
String name()	駒の名前を返す
void move(int, int)	駒を特定の座標に移動させる
ArrayList¡NextMove¿ movableList(int[ ][ ])	駒の移動可能な座標を判定しリストを返す

5　実験結果

ごろごろ動物将棋が先手有利か後手有利かを検証するために, 本研究で作成したプログラムで AI 同士の対戦を先読み手数を 4 手として 300 回行った．AI 同士の対戦結果を表 8 に示す．持ち駒なしなので千日手が多く発生し引き分けが多い結果となったが，表 8 の結果から先手が有利だと推測される．

以下では先手有利と言えるかどうかを統計的に検証する．

表 8　AI 同士の対戦結果 (試行回数 300 回)

先手勝ち	後手勝ち	引き分け
132	90	78

勝率 p の勝負を N 回行ったときの標準偏差 s は以下の式で与えられる．

$$s = \sqrt{N * p * (1 - p)}$$

表 8 の結果から引き分けが 26% ほどあるので，勝ち 37%, 負け 37%, 引き分け 26% と仮定する．$p = 0.37$ と仮定すると，$N = 300$ の場合標準偏差 s_{300} は

$$s_{300} = \sqrt{300 * 0.37 * 0.63} = 8.36$$

となる．統計学によると, 信頼度 95 %となるのは標準偏差の 1.96 倍の区間なので, 勝率 37 %負率 37 %の勝負を 300 回行なった場合の勝ち数, 負け数は 95 %の確率で 111±8.36*1.96 の範囲, つまり 95 回 ∼127 回に収まる．

よって, 勝ち数, 負け数が共に上記の範囲に収まらず, 勝率と負率が同じであるという仮定に対して統計上有意に勝ち数は高く，負け数は低くなる．したがって，先手有利だと推測される．

6　結論・今後の課題

本研究ではごろごろ動物将棋の解析のため, AI 同士の対戦, 持ち駒なしの不完全なごろごろ動物将棋プログラムを作成した．AI 同士の対戦の結果から持駒無しの場合は先手有利と分かったが, 先手必勝である根拠は得ることは出来なかったのである．

今後の課題としてごろごろ動物将棋のルールに基づいて持ち駒の仕組みの追加や, アプリケーションの高速化, 適切な評価値の計算方法を見つけることが完全解析をするために必要なことだと考えられる．ただし, 完全解析されている動物将棋の総局面数が 1,567,925,964 通りである [1] ため, より複雑なごろごろ動物将棋の総局面数は $1.0\text{x}10^{22}$ 通りであるので完全解析は困難であると予測される．完全解析への対処方法として, 自駒が相手駒より少ないときは自身のライオンに近いほど高評価とし, 多ければ敵陣へ進める手を高評価とするような戦略性を加えることが大切だと考えられる．また, 本将棋では金将を守りに, 銀将を攻めに使う戦略が多く存在し, それに倣い犬は自ライオンに近いほど高評価, 猫は相手ライオンに近いほど高評価にするよう評価システムを変えることや, 対戦データベースを保存し勝ちパターンを分析することが方法として考えられる．また, 本研究で作成したプログラムに対人戦が出来るようにすることも今後の課題の 1 つだと考えられる．

謝辞

本研究で石水先生には大変お世話になりました.
ありがとうございました.

参考文献

[1] 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/

[2] 塩田好, 石水隆, 山本博史: 「アンパンマンはじめてしょうぎ」の完全解析, 情報処理学会関西支部 支部大会講演論文集 (2013).http://id.nii.ac.jp/1001/00096792/

[3] シルバースタージャパン：みんなのどうぶつしょうぎ，Nintendo Switch 用ソフト，(2019) https://www.silverstar.co.jp/02products/dobutsushogi/switch/

[4] シルバースタージャパン：みんなのどうぶつしょうぎ，PlayStation4 用ソフト，(2019) https://www.silverstar.co.jp/02products/dobutsushogi/ps4/

[5] 株式会社ねこまどぶろぐ：11 月下旬「ごろごろどうぶつしょうぎ」発売決定!(2012) http://nekomadoblog.jugem.jp/?eid=489

[6] 北尾まどか，川崎智秀，藤田麻衣子：どうぶつしょうぎ Q 1 キャッチと王手，幻冬舎エデュケーション，(2011)

[7] 北尾まどか，川崎智秀，藤田麻衣子：どうぶつしょうぎ Q 2 王手と詰み，幻冬舎エデュケーション，(2012)

付録 A　ソースプログラム

GorogoroSyogi クラス

以下に第 4.2 章で説明した GorogoroSyogi クラスのソースコードを示す.

```
1     package syogi;
2
3   import java.util.Scanner;
4   import java.io.*;
5
6   public class gorogoroSyogi {
7       //  先手の駒  盤面手前
8       final static int LION = 1; // ライオン
9       final static int DOG1 = 2; // 犬 1
10      final static int DOG2 = 3; // 犬  2
11      final static int NEKO1 = 4; // 猫 1
12      final static int NEKO2 = 5; // 猫 2
13      final static int HIYOKO1 = 6; // ひよこ 1
14      final static int HIYOKO2 = 7; // ひよこ 2
15      final static int HIYOKO3 = 8; // ひよこ 3
16
17      final static int CAT1 = 9;  // パワーアップ猫 1
18      final static int CAT2 = 10; // パワーアップ猫 2
19      final static int BIRD1 = 11; // 鶏 1
20      final static int BIRD2 = 12; // 鶏 2
21      final static int BIRD3 = 13; // 鶏 3
22
23      // 後手の駒 盤面奥
24      final static int E_LION = -1; // ライオン
25      final static int E_DOG1 = -2; // 犬 1
26      final static int E_DOG2 = -3; // 犬  2
27      final static int E_NEKO1 = -4; // 猫 1
28      final static int E_NEKO2 = -5; // 猫 2
29      final static int E_HIYOKO1 = -6; // ひよこ 1
30      final static int E_HIYOKO2 = -7; // ひよこ 2
31      final static int E_HIYOKO3 = -8; // ひよこ 3
32
33      final static int E_CAT1 = -9; // パワーアップ猫 1
34      final static int E_CAT2 = -10; // パワーアップ猫 2
35      final static int E_BIRD1 = -11; // 鶏 1
36      final static int E_BIRD2 = -12; // 鶏 2
37      final static int E_BIRD3 = -13; // 鶏 3
38
39      final static int EMPTY = 0; // 空白
40      final static int BORDER = Integer.MAX_VALUE; // 盤外
41      // 棋譜出力用
42      static FileWriter pass, rPass;
43      static PrintWriter fp, rfp;
44
45      public static void main(String[] args) {
46           Board board = new Board(); // 駒の情報を登録する
```

```
47              String score; // 棋譜出力用
48              FileWriter pass = null;
49              PrintWriter fp = null;
50              Scanner keyBoardScanner = new Scanner(System.in);
51              System.out.println("先読み数を入力してください");
52              String depth = keyBoardScanner.next(); // 先読み値数
53              board.setMaxDepth(Integer.parseInt(depth));
54              int count = 0; // 連続試行回数用カウント
55              int first = 0, second = 0, draw = 0;
56
57              try{
58                 pass = new FileWriter("gorogoroSyogiScore.txt");
59                 fp = new PrintWriter(pass);
60              }catch(IOException e){
61                 System.err.println(e);
62              }
63
64              while(count < 1) { // 1 なら 1 回，100 なら 100 回試合を行う
65                 while(true){
66                     board.showBoard(); // 盤面出力
67                     board.createMovableList(0); // 先手の移動可能な手を設定する
68                     if(board.isChecked(0)){ // 王手の判定
69                         System.out.println("王手");
70                     }
71                     if(board.checkWin(1)){ // 詰みの判定
72                         break;
73                     }
74                     score = board.com(0); // 先手の駒を移動する
75                     fp.print(score); // 棋譜出力
76                     board.showBoard(); // 盤面出力
77                     board.createMovableList(1); // 後手の移動可能な手を決める
78                     if(board.isChecked(1)){ // 王手の判定
79                         System.out.println("王手");
80                     }
81                     if(board.checkWin(0)){ // 詰みの判定
82                         break;
83                     }
84                     score = board.com(1);  // 後手の駒を移動する
85
86                     fp.print(score); // 棋譜出力
87                 }
88                 fp.close();
89                 count += 1;
90                 if(board.winner() == 1) { // 先手勝ち
91                     first += 1;
92                 }else if(board.winner() == -1) { // 後手勝ち
93                     second += 1;
94                 }else { // 引き分け
95                     draw += 1;
96                 }
97                 board.resetMoves(); // 手数のリセット
98                 board = new Board(); // 初期盤面を生成
99                 score = null; // 棋譜をリセット
```

```
100             }
101             System.out.println("試行回数"+count+"回");
102             System.out.println("先手勝ち:"+first+"回");
103             System.out.println("後手勝ち:"+second+"回");
104             System.out.println("引き分け:"+draw+"回");
105         }
106     }
107
```

Boardクラス

以下に第 4.3 章で説明した Board クラスのソースコードを示す.

```
1     package syogi;
2
3   import java.util.ArrayList;
4
5   import java.util.Random;
6   import syogi.NextMove;
7   import syogi.Piece;
8
9   public class Board {
10      //  先手の駒  盤面手前
11      final static int LION = 1; // ライオン
12      final static int DOG1 = 2; // 犬 1
13      final static int DOG2 = 3; // 犬  2
14      final static int NEKO1 = 4; // 猫 1
15      final static int NEKO2 = 5; // 猫 2
16      final static int HIYOKO1 = 6; // ひよこ 1
17      final static int HIYOKO2 = 7; // ひよこ 2
18      final static int HIYOKO3 = 8; // ひよこ 3
19
20      final static int CAT1 = 9; // パワーアップ猫 1
21      final static int CAT2 = 10; // パワーアップ猫 2
22      final static int BIRD1 = 11; // 鶏 1
23      final static int BIRD2 = 12; // 鶏 2
24      final static int BIRD3 = 13; // 鶏 3
25
26      // 後手の駒 盤面奥
27      final static int E_LION = -1; // ライオン
28      final static int E_DOG1 = -2; // 犬 1
29      final static int E_DOG2 = -3; // 犬  2
30      final static int E_NEKO1 = -4; // 猫 1
31      final static int E_NEKO2 = -5; // 猫 2
32      final static int E_HIYOKO1 = -6; // ひよこ 1
33      final static int E_HIYOKO2 = -7; // ひよこ 2
34      final static int E_HIYOKO3 = -8; // ひよこ 3
35
36      final static int E_CAT1 = -9; // パワーアップ猫 1
37      final static int E_CAT2 = -10; // パワーアップ猫 2
38      final static int E_BIRD1 = -11; // 鶏 1
39      final static int E_BIRD2 = -12; // 鶏 2
40      final static int E_BIRD3 = -13; // 鶏 3
41
```

```
42      final static int EMPTY = 0; // 空白
43      final static int BORDER = Integer.MAX_VALUE; // 盤外
44      final static boolean isChessStyleScore = false; // 棋譜表記をチェス式か将棋式
  か
45
46      // 将棋盤面
47      public int[][] board = {
48              {BORDER, BORDER, BORDER, BORDER, BORDER, BORDER, BORDER},
49              {BORDER, E_NEKO1, E_DOG1, E_LION, E_DOG2, E_NEKO2, BORDER},
50              {BORDER, EMPTY, EMPTY, EMPTY, EMPTY, EMPTY, EMPTY, BORDER},
51              {BORDER, EMPTY, E_HIYOKO1, E_HIYOKO2, E_HIYOKO3, EMPTY, BORDER},
52              {BORDER, EMPTY, HIYOKO1, HIYOKO2, HIYOKO3, EMPTY, BORDER},
53              {BORDER, EMPTY, EMPTY, EMPTY, EMPTY, EMPTY, EMPTY, BORDER},
54              {BORDER, NEKO1, DOG1, LION, DOG2, NEKO2, BORDER},
55              {BORDER, BORDER, BORDER, BORDER, BORDER, BORDER, BORDER}
56      };
57      int nextFile; // 移動先の X 座標
58      int nextRank; // 移動先の Y 座標
59
60      // 駒
61      Piece lion, dog1, dog2, neko1, neko2, hiyoko1, hiyoko2, hiyoko3, cat1, cat2, bird1, bird
62            e_lion, e_dog1, e_dog2, e_neko1, e_neko2, e_hiyoko1, e_hiyoko2, e_hiyoko3, e_cat1,
63      boolean resign = false; // リタイアしたか
64      boolean checkmate = false; // 詰みかどうか（チェックメイト）
65      boolean stalemate = false; // 詰みかどうか（スティルメイト）
66      ArrayList<NextMove> movableList; // 候補手のリスト
67
68      int maxDepth = 0; // 先読みする手数の上限
69      boolean doResign = false; // 負けが確定したときにリタイアするかどうか
70      static int moves = 0; // 手数
71      static int lookAheadMoves = 0; // 先読みの手数
72      final static int maxMove = 500; // 手数の上限
73      static int[][][] boardRecord = new int [maxMove][6][5]; // 盤面記録用
74      int winner = 0; // 先手勝ち:1, 後手勝ち:-1, 引き分け:0
75      int pieceValue = 0; // 駒を取った時、評価をプラスする
76
77      /*
78       * コンストラクタ
79       *  駒の初期設定
80       */
81      public Board() {
82          // 駒のインスタンス生成
83          lion = new Piece(LION);
84          dog1 = new Piece(DOG1);
85          dog2 = new Piece(DOG2);
86          neko1 = new Piece(NEKO1);
87          neko2 = new Piece(NEKO2);
88          hiyoko1 = new Piece(HIYOKO1);
89          hiyoko2 = new Piece(HIYOKO2);
90          hiyoko3 = new Piece(HIYOKO3);
91
92          cat1 = null;
93          cat2 = null;
```

```
94          bird1 = null;
95          bird2 = null;
96          bird3 = null;
97
98          e_lion = new Piece(E_LION);
99          e_dog1 = new Piece(E_DOG1);
100         e_dog2 = new Piece(E_DOG2);
101         e_neko1 = new Piece(E_NEKO1);
102         e_neko2 = new Piece(E_NEKO2);
103         e_hiyoko1 = new Piece(E_HIYOKO1);
104         e_hiyoko2 = new Piece(E_HIYOKO2);
105         e_hiyoko3 = new Piece(E_HIYOKO3);
106
107         e_cat1 = null;
108         e_cat2 = null;
109         e_bird1 = null;
110         e_bird2 = null;
111         e_bird3 = null;
112     }
113
114     /**
115      * コンストラクタ
116      * 盤面に駒を配置
117      * @param int[][] board 盤面
118      */
119     public Board(int[][] board) {
120         // 盤面を更新する
121         for(int i = 1; i <= 6; ++i) {
122             for(int j = 1; j <= 5; ++j) {
123                 this.board[i][j] = board[i][j];
124             }
125         }
126         // 盤面にある駒を再生成する
127         for(int i = 1; i <= 6; ++i) {
128             for(int j = 1; j <= 5; ++j) {
129                 switch(board[i][j]) {
130                 case LION:
131                     lion = new Piece(LION, j, i);
132                     break;
133                 case DOG1:
134                     dog1 = new Piece(DOG1, j, i);
135                     break;
136                 case DOG2:
137                     dog2 = new Piece(DOG2, j, i);
138                     break;
139                 case NEKO1:
140                     neko1 = new Piece(NEKO1, j, i);
141                     break;
142                 case NEKO2:
143                     neko2 = new Piece(NEKO2, j, i);
144                     break;
145                 case HIYOKO1:
146                     hiyoko1 = new Piece(HIYOKO1, j, i);
```

```
147                     break;
148                 case HIYOKO2:
149                     hiyoko2 = new Piece(HIYOKO2, j, i);
150                     break;
151                 case HIYOKO3:
152                     hiyoko3 = new Piece(HIYOKO3, j, i);
153                     break;
154                 case CAT1:
155                     cat1 = new Piece(CAT1, j, i);
156                     break;
157                 case CAT2:
158                     cat2 = new Piece(CAT2, j, i);
159                     break;
160                 case BIRD1:
161                     bird1 = new Piece(BIRD1, j, i);
162                     break;
163                 case BIRD2:
164                     bird2 = new Piece(BIRD2, j, i);
165                     break;
166                 case BIRD3:
167                     bird3 = new Piece(BIRD3, j, i);
168                     break;
169                 case E_LION:
170                     e_lion = new Piece(E_LION, j, i);
171                     break;
172                 case E_DOG1:
173                     e_dog1 = new Piece(E_DOG1, j, i);
174                     break;
175                 case E_DOG2:
176                     e_dog2 = new Piece(E_DOG2, j, i);
177                     break;
178                 case E_NEKO1:
179                     e_neko1 = new Piece(E_NEKO1, j, i);
180                     break;
181                 case E_NEKO2:
182                     e_neko2 = new Piece(E_NEKO2, j, i);
183                     break;
184                 case E_HIYOKO1:
185                     e_hiyoko1 = new Piece(E_HIYOKO1, j, i);
186                     break;
187                 case E_HIYOKO2:
188                     e_hiyoko2 = new Piece(E_HIYOKO2, j, i);
189                     break;
190                 case E_HIYOKO3:
191                     e_hiyoko3 = new Piece(E_HIYOKO3, j, i);
192                     break;
193                 case E_CAT1:
194                     e_cat1 = new Piece(E_CAT1, j, i);
195                     break;
196                 case E_CAT2:
197                     e_cat2 = new Piece(E_CAT2, j, i);
198                     break;
199                 case E_BIRD1:
```

```
200                     e_bird1 = new Piece(E_BIRD1, j, i);
201                     break;
202                 case E_BIRD2:
203                     e_bird2 = new Piece(E_BIRD2, j, i);
204                     break;
205                 case E_BIRD3:
206                     e_bird3 = new Piece(E_BIRD3, j, i);
207                     break;
208                 }
209             }
210         }
211     }
212
213     /**
214      * 盤面を表示する
215      */
216     public void showBoard() {
217         System.out.println("   1 2 3 4 5");
218         for(int i = 0; i < board.length; ++i) {
219             switch(i) {
220             case 0:
221                 System.out.print("  ");
222                 break;
223             case 1:
224                 System.out.print("一");
225                 break;
226             case 2:
227                 System.out.print("二");
228                 break;
229             case 3:
230                 System.out.print("三");
231                 break;
232             case 4:
233                 System.out.print("四");
234                 break;
235             case 5:
236                 System.out.print("五");
237                 break;
238             case 6:
239                 System.out.print("六");
240                 break;
241             case 7:
242                 System.out.print("  ");
243                 break;
244             }
245             for(int j = 0; j < board[i].length; ++j) {
246                 System.out.print(showPiece(board[i][j]));
247             }
248             System.out.println();
249         }
250     }
251
252     /**
```

```
253         * 駒を表示する
254         * @param type 駒の種類
255         * @return   駒の文字列
256         */
257        public String showPiece(int type) {
258            switch(type) {
259            case LION:
260                return "ら";
261            case DOG1:
262            case DOG2:
263                return "い";
264            case NEKO1:
265            case NEKO2:
266                return "ね";
267            case HIYOKO1:
268            case HIYOKO2:
269            case HIYOKO3:
270                return "ひ";
271            case CAT1:
272            case CAT2:
273                return "き";
274            case BIRD1:
275            case BIRD2:
276            case BIRD3:
277                return "と";
278
279            case E_LION:
280                return "ラ";
281            case E_DOG1:
282            case E_DOG2:
283                return "イ";
284            case E_NEKO1:
285            case E_NEKO2:
286                return "ネ";
287            case E_HIYOKO1:
288            case E_HIYOKO2:
289            case E_HIYOKO3:
290                return "ヒ";
291            case E_CAT1:
292            case E_CAT2:
293                return "キ";
294            case E_BIRD1:
295            case E_BIRD2:
296            case E_BIRD3:
297                return "ト";
298            case EMPTY:
299                return "  ";
300            case BORDER:
301                return " ■";
302            default:
303                return "?";
304            }
305        }
```

```
306
307         /**
308          * COM の番
309          * maxDepth で指定した手数を先読みし、最良の手を指す
310          * @param playerNum プレイヤー番号
311          * @return 指した手の棋譜
312          */
313         public String com(int playerNum) {
314             int type, nextFile, nextRank; // 移動する駒の種類, 移動先の座標
315             Piece piece = null; // 移動する駒
316             int bestValue = 0; // 最も良い盤面の評価値
317             NextMove bestMove = null; // 最も良い手
318             int nextPlayerNum; // 次の手番プレイヤー
319 
320             if(movableList.size() == 0) { // 移動できる候補手がない場合
321                 System.out.println("リタイアします");
322                 resign = true;
323                 if(isChessStyleScore) {
324                     return (playerNum == 0) ? "1-0" : "0-1";
325                 }else {
326                     return "リタイア";
327                 }
328             }else if(movableList.size() == 1) { // 候補手が 1 つしかない場合
329                 bestMove = movableList.get(0); // 評価値を求めない
330             }else {
331                 if(playerNum == 0) { // 先手の場合、評価値が高いほど良い手とする
332                     nextPlayerNum = 1; // 次の手番のプレイヤー番号
333                     bestValue = Integer.MIN_VALUE;
334                     bestMove = movableList.get(0);
335                     ++lookAheadMoves; // 先読みのため手数を＋１する
336                     for(int i = 0; i < movableList.size(); ++i) {
337                         NextMove nextMove = movableList.get(i); // i 番目の候補手
338                         Board nextBoard = nextBoard(nextMove, nextPlayerNum); // 次
    の盤面を生成
339                         int value = nextBoard.value(nextPlayerNum, maxDepth); // 盤
    面の評価値を計算する
340                         if(value > bestValue) { // 評価値が高い手のとき
341                             bestMove = nextMove; // 高評価の手を代入
342                             bestValue = value; // 評価値を代入
343                         }
344                         if(bestValue == Integer.MAX_VALUE) { // 評価値が Max の場合
345                             break; // ループ終了
346                         }
347                     }
348                     --lookAheadMoves; // 先読みが終わり手数を１つ戻す
349                     if(bestValue == Integer.MIN_VALUE && doResign) { // 評価値が最
    小の場合、負けが確定
350                         System.out.println("リタイアします");
351                         resign = true;
352                         if(isChessStyleScore) {
353                             return "1-0";
354                         }else {
355                             return "リタイア";
```

```
356                         }
357                     }
358                 }else { // 後手の場合、評価値が低いほど良い手とみなす
359                     nextPlayerNum = 0; // 次の手番のプレイヤー番号
360                     bestValue = Integer.MAX_VALUE;
361                     bestMove = movableList.get(0);
362                     ++lookAheadMoves; // 先読みのために手数を 1 増やす
363                     for(int i = 0; i <  movableList.size(); ++i) {
364                         NextMove nextMove = movableList.get(i); // i 番目の候補手
365                         Board nextBoard = nextBoard(nextMove, nextPlayerNum); // 盤面の評価値を計算
366                         int value = nextBoard.value(nextPlayerNum, maxDepth); // 盤面の評価値を計算する
367                         if(value < bestValue) { // 低評価の手を見つけたとき
368                             bestMove = nextMove; // 低評価の手を代入
369                             bestValue = value; // 評価値を代入
370                         }
371                         if(bestValue == Integer.MIN_VALUE) { // ;評価値が最小の場合、必勝の手とする
372                             --lookAheadMoves; // 先読みが終了し、手数を 1 つ戻す
373                             break; // ループ終了
374                         }
375                     }
376                     --lookAheadMoves; // 先読みが終了し、手数を 1 つ戻す
377                     if(bestValue == Integer.MAX_VALUE && doResign) { // 評価値が最大の場合、負けが確定
378                         System.out.println("リタイアします");
379                         resign = true;
380                         if(isChessStyleScore) {
381                             return "0-1";
382                         }else {
383                             return "リタイア";
384                         }
385                     }
386                 }
387             }
388             type = bestMove.type(); // 移動する駒の種類
389             nextFile = bestMove.nextFile(); // 移動先の X 座標
390             nextRank = bestMove.nextRank(); // 移動先の Y 座標
391             // 動かす駒を設定する
392             switch (type) {
393             case LION : piece = lion; break;
394             case DOG1 : piece = dog1; break;
395             case DOG2 : piece = dog2; break;
396             case NEKO1 : piece = neko1; break;
397             case NEKO2 : piece = neko2; break;
398             case HIYOKO1 : piece = hiyoko1; break;
399             case HIYOKO2 : piece = hiyoko2; break;
400             case HIYOKO3 : piece = hiyoko3; break;
401             case CAT1 : piece = cat1; break;
402             case CAT2 : piece = cat2; break;
403             case BIRD1 : piece = bird1; break;
404             case BIRD2 : piece = bird2; break;
```

```
405             case BIRD3 : piece = bird3; break;
406             case E_LION : piece = e_lion; break;
407             case E_DOG1 : piece = e_dog1; break;
408             case E_DOG2 : piece = e_dog2; break;
409             case E_NEKO1 : piece = e_neko1; break;
410             case E_NEKO2 : piece = e_neko2; break;
411             case E_HIYOKO1 : piece = e_hiyoko1; break;
412             case E_HIYOKO2 : piece = e_hiyoko2; break;
413             case E_HIYOKO3 : piece = e_hiyoko3; break;
414             case E_CAT1 : piece = e_cat1; break;
415             case E_CAT2 : piece = e_cat2; break;
416             case E_BIRD1 : piece = e_bird1; break;
417             case E_BIRD2 : piece = e_bird2; break;
418             case E_BIRD3 : piece = e_bird3; break;
419             }
420             String score = movePiece(piece, type, nextFile, nextRank); // 駒を移動
    させる
421             System.out.println(moves + ":" + score + "[" + bestValue + "]");
422             ++moves;
423             return score;
424         }
425
426
427         /**
428          * 指定した位置に駒を移動させ、棋譜を返す
429          * @param Piece piece 移動させる駒
430          * @param int type 移動させる駒の種類
431          * @param int nextFile  移動先の X 座標
432          * @param int nextRank  移動先の Y 座標
433          * @param int playerNum プレイヤー番号
434          */
435         public String movePiece(Piece piece, int type, int nextFile, int nextRank) {
436             String score; // 棋譜
437             if(isChessStyleScore) { // チェス風の棋譜を作成
438                 switch (type) {
439                 case LION : score = "L"; break;
440                 case DOG1 : score = "D1"; break;
441                 case DOG2 : score = "D2"; break;
442                 case NEKO1 : score = "N1"; break;
443                 case NEKO2 : score = "N2"; break;
444                 case HIYOKO1 : score = "H1"; break;
445                 case HIYOKO2 : score = "H2"; break;
446                 case HIYOKO3 : score = "H3"; break;
447                 case CAT1 : score = "C1"; break;
448                 case CAT2 : score = "C2"; break;
449                 case BIRD1 : score = "B1"; break;
450                 case BIRD2 : score = "B2"; break;
451                 case BIRD3 : score = "B3"; break;
452                 case E_LION : score ="EL"; break;
453                 case E_DOG1 : score = "ED1"; break;
454                 case E_DOG2 : score = "ED2"; break;
455                 case E_NEKO1 : score = "EN1"; break;
456                 case E_NEKO2 : score = "EN2"; break;
```

```
457             case E_HIYOKO1 : score = "EH1"; break;
458             case E_HIYOKO2 : score ="EH2"; break;
459             case E_HIYOKO3 : score = "EH3"; break;
460             case E_CAT1 : score = "EC1"; break;
461             case E_CAT2 : score = "EC2"; break;
462             case E_BIRD1 : score = "EB1"; break;
463             case E_BIRD2 : score = "EB2"; break;
464             case E_BIRD3 : score = "EB3"; break;
465             default : score = "?"; break;
466             }
467             if (board[nextRank][nextFile] != EMPTY) score += "x";
468             switch (nextFile) {
469             case 1 : score += "a"; break;
470             case 2 : score += "b"; break;
471             case 3 : score += "c"; break;
472             case 4 : score += "d"; break;
473             case 5 : score += "e"; break;
474             default : score += "?"; break;
475             }
476             switch (nextRank) {
477             case 1 : score += "1 "; break;
478             case 2 : score += "2 "; break;
479             case 3 : score += "3 "; break;
480             case 4 : score += "4 "; break;
481             case 5 : score += "5 "; break;
482             case 6 : score += "6 "; break;
483             default : score += "? " ; break;
484             }
485         }else { // 将棋風の棋譜を生成
486             switch (nextFile) {
487             case 1 : score = "１"; break;
488             case 2 : score = "２"; break;
489             case 3 : score = "３"; break;
490             case 4 : score = "４"; break;
491             case 5 : score = "５"; break;
492             default : score = "？"; break;
493             }
494             switch (nextRank) {
495             case 1 : score += "一"; break;
496             case 2 : score += "二"; break;
497             case 3 : score += "三"; break;
498             case 4 : score += "四"; break;
499             case 5 : score += "五"; break;
500             case 6 : score += "六"; break;
501             default : score += "？" ; break;
502             }
503             switch (type) {
504             case LION : score += "ら   "; break; //ライオン
505             case DOG1 : score += "い_1 "; break; //犬１
506             case DOG2 : score += "い_2 "; break; //犬２
507             case NEKO1 : score += "ね_1 "; break; //猫１
508             case NEKO2 : score += "ね_2 "; break; //猫２
509             case HIYOKO1 : score += "ひ_1 "; break; //ひよこ１
```

```
510             case HIYOKO2 : score += "ひ_2 "; break; //ひよこ２
511             case HIYOKO3 : score += "ひ_3 "; break; //ひよこ３
512             case CAT1 : score += "き_1 "; break; //成り猫１
513             case CAT2 : score += "き_2 "; break; //成り猫２
514             case BIRD1 : score += "と_1 "; break; //成りひよこ１
515             case BIRD2 : score += "と_2 "; break; //成りひよこ２
516             case BIRD3 : score += "と_3 "; break; //成りひよこ３
517             case E_LION : score += "ラ  "; break; //ライオン
518             case E_DOG1 : score += "イ_1 "; break; //犬１
519             case E_DOG2 : score += "イ_2 "; break; //犬２
520             case E_NEKO1 : score += "ネ_1 "; break; //猫１
521             case E_NEKO2 : score += "ネ_2 "; break; //猫２
522             case E_HIYOKO1 : score += "ヒ_1 "; break; //ひよこ１
523             case E_HIYOKO2 : score += "ヒ_2 "; break; //ひよこ２
524             case E_HIYOKO3 : score += "ヒ_3 "; break; //ひよこ３
525             case E_CAT1 : score += "キ_1 "; break; //成り猫１
526             case E_CAT2 : score += "キ_2 "; break; //成り猫２
527             case E_BIRD1 : score += "ト_1 "; break; //成りひよこ１
528             case E_BIRD2 : score += "ト_2 "; break; //成りひよこ２
529             case E_BIRD3 : score += "ト_3 "; break; //成りひよこ３
530             default : score += "？ "; break;
531             }
532         }
533         if (board[nextRank][nextFile] != EMPTY) { // 移動先に駒がある場合
534             removePiece (nextFile, nextRank); // 移動先にある駒を取り除く
535         }
536         board[piece.rank()][piece.file()] = EMPTY; // 移動前のマスを空白に
537         piece.move (nextFile, nextRank); // 駒を移動
538         board[piece.rank()][piece.file()] = type; // 移動後のマスを指定した駒に
539 
540         // ひよこの成り
541         promotePiece(board);
542 
543         recordBoard(); // 移動後の盤面を記録する
544         return score;
545     }
546 
547 
548     public void removePiece(int nextFile, int nextRank) {
549         switch (board [nextRank][nextFile]) {
550         case LION : // 移動先にライオン
551             lion = null; // ライオンを取り除く
552             break;
553         case DOG1 : // 移動先に犬_1
554             dog1 = null; // 犬_1 を取り除く
555             break;
556         case DOG2 : // 移動先に犬_2
557             dog2 = null; // 犬_2 を取り除く
558             break;
559         case NEKO1 : // 移動先に猫_1
560             neko1 = null; // 猫_1 を取り除く
561             break;
562         case NEKO2 : // 移動先に猫_2
```

```
563             neko2 = null; // 猫_2 を取り除く
564             break;
565         case HIYOKO1 : // 移動先にひよこ_1
566             hiyoko1 = null; // ひよこ_1 を取り除く
567             break;
568         case HIYOKO2 : // 移動先にひよこ_2
569             hiyoko2 = null; // ひよこ_2 を取り除く
570             break;
571         case HIYOKO3 : // 移動先にひよこ_3
572             hiyoko3 = null; // ひよこ_3 を取り除く
573             break;
574         case CAT1 : // 移動先に成り猫_1
575             cat1 = null; // 成り猫_1 を取り除く
576             break;
577         case CAT2 : // 移動先に成り猫_2
578             cat2 = null; // 成り猫_2 を取り除く
579             break;
580         case BIRD1 : // 移動先に成りひよこ_1
581             bird1 = null; // 成りひよこ_1 を取り除く
582             break;
583         case BIRD2 : // 移動先に成りひよこ_2
584             bird2 = null; // 成りひよこ_2 を取り除く
585             break;
586         case BIRD3 : // 移動先に成りひよこ_3
587             bird3 = null; // 成りひよこ_3 を取り除く
588             break;
589
590         case E_LION : // 移動先に E-ライオン
591             e_lion = null; // E-ライオンを取り除く
592             break;
593         case E_DOG1 : // 移動先に E-犬_1
594             e_dog1 = null; // E-犬_1 を取り除く
595             break;
596         case E_DOG2 : // 移動先に E=犬_2
597             e_dog2 = null; // E-犬_2 を取り除く
598             break;
599         case E_NEKO1 : // 移動先に E-猫_1
600             e_neko1 = null; // E-猫_1 を取り除く
601             break;
602         case E_NEKO2 : // 移動先に E-猫_2
603             e_neko2 = null; // E-猫_2 を取り除く
604             break;
605         case E_HIYOKO1 : // 移動先に E-ひよこ_1
606             e_hiyoko1 = null; // E-ひよこ_1 を取り除く
607             break;
608         case E_HIYOKO2 : // 移動先に E-ひよこ_2
609             e_hiyoko2 = null; // E-ひよこ_2 を取り除く
610             break;
611         case E_HIYOKO3 : // 移動先に E-ひよこ_3
612             e_hiyoko3 = null; // E-ひよこ_3 を取り除く
613             break;
614         case E_CAT1 : // 移動先に E-成り猫_1
615             e_cat1 = null; // E_成り猫_1 を取り除く
```

```
616                 break;
617             case E_CAT2 : // 移動先に E-成り猫_2
618                 e_cat2 = null; // E-成り猫_2 を取り除く
619                 break;
620             case E_BIRD1 : // 移動先に E-成りひよこ_1
621                 e_bird1 = null; // E-成りひよこ_1 を取り除く
622                 break;
623             case E_BIRD2 : // 移動先に E-成りひよこ_2
624                 e_bird2 = null; // E-成りひよこ_2 を取り除く
625                 break;
626             case E_BIRD3 : // 移動先に E-成りひよこ_3
627                 e_bird3 = null; // E-成りひよこ_3 を取り除く
628                 break;
629             }
630         }
631 
632         /**
633          * 駒を成る
634          * @param board 移動後の盤面
635          */
636         public void promotePiece(int[][] board) {
637             for(int i = 1; i <= 2; i++) { // 1,2 行にひよこ、猫がいる場合
638                 for(int j = 1; j <= 6; j++) {
639                     if(board[i][j] == HIYOKO1) {
640                         hiyoko1 = null; // ひよこ 1 を削除する
641                         board[i][j] = BIRD1; // 成りひよこ 1 を置く
642                         bird1 = new Piece(BIRD1, j, i);
643                     }else if(board[i][j] == HIYOKO2) {
644                         hiyoko2 = null; // ひよこ 2 を削除する
645                         board[i][j] = BIRD2; // 成りひよこ 2 を置く
646                         bird2 = new Piece(BIRD2, j, i);
647                     }else if(board[i][j] == HIYOKO3) {
648                         hiyoko3 = null; // ひよこ 3 を削除する
649                         board[i][j] = BIRD3; // 成りひよこ 3 を置く
650                         bird3 = new Piece(BIRD3, j, i);
651                     }else if(board[i][j] == NEKO1) {
652                         neko1 = null; // 猫 1 を削除する
653                         board[i][j] = CAT1; // 成り猫 1 を置く
654                         cat1 = new Piece(CAT1, j, i);
655                     }else if(board[i][j] == NEKO2) {
656                         neko2 = null; // 猫 2 を削除する
657                         board[i][j] = CAT2; // 成り猫 2 を置く
658                         cat2 = new Piece(CAT2, j, i);
659                     }
660                 }
661             }
662             for(int i = 5; i <= 6; i++) { // 5,6 行に E-ひよこ、E-猫がいる場合
663                 for(int j = 1; j <= 6; j++) {
664                     if(board[i][j] == E_HIYOKO1) {
665                         e_hiyoko1 = null; // ひよこ 1 を削除する
666                         board[i][j] = E_BIRD1; // 成りひよこ 1 を置く
667                         e_bird1 = new Piece(E_BIRD1, j, i);
668                     }else if(board[i][j] == E_HIYOKO2) {
```

```
669                     e_hiyoko2 = null; // ひよこ２を削除する
670                     board[i][j] = E_BIRD2; // 成りひよこ２を置く
671                     e_bird2 = new Piece(E_BIRD2, j, i);
672                 }else if(board[i][j] == E_HIYOKO3) {
673                     e_hiyoko3 = null; // ひよこ３を削除する
674                     board[i][j] = E_BIRD3; // 成りひよこ３を置く
675                     e_bird3 = new Piece(E_BIRD3, j, i);
676                 }else if(board[i][j] == E_NEKO1) {
677                     e_neko1 = null; // 猫１を削除する
678                     board[i][j] = E_CAT1; // 成り猫１を置く
679                     e_cat1 = new Piece(E_CAT1, j, i);
680                 }else if(board[i][j] == E_NEKO2) {
681                     e_neko2 = null; // 猫２を削除する
682                     board[i][j] = E_CAT2; // 成り猫２を置く
683                     e_cat2 = new Piece(E_CAT2, j, i);
684                 }
685             }
686         }
687     }
688
689     /**
690      * 決着したかどうか
691      * @param int playerNum プレイヤー番号
692      * @return 勝敗
693      */
694     public boolean checkWin(int playerNum) {
695         if (playerNum == 0) { // 先手の手番
696             if (e_lion == null) { // E-ライオンを取った
697                 System.out.println ("先手の勝利！");
698                 winner = 1;
699                 return true;
700             }
701             else if (isMate (1)) { // E-ライオンが詰んだ
702                 if (isChecked (1)) { // E-ライオンに王手がかかっている
703                     System.out.println ("チェックメイト！");
704                     System.out.println ("先手の勝利！");
705                     winner = 1;
706                 }
707                 return true;
708             } else if (resign) { // 投了した
709                 System.out.println ("後手の勝利！");
710                 winner = -1;
711                 return true;
712             } else if (checkPerpetual()) {
713                 System.out.println ("千日手");
714                 System.out.println ("引き分けです");
715                 winner = 0;
716                 return true;
717             } else {
718                 return false;
719             }
720         }else { // 後手の手番
721             if(lion == null) { // ライオンを取った場合
```

```
722                     System.out.println("後手の勝利!");
723                     winner = -1;
724                     return true;
725                 }
726                 else if (isMate (0)) { // 先手が詰んだ
727                     if (isChecked (0)) { // 先手に王手がかかっている
728                         System.out.println ("チェックメイト！");
729                         System.out.println ("後手の勝利！");
730                         winner = -1;
731                     }
732                     return true;
733                 } else if (resign) { // 投了した
734                     System.out.println ("先手の勝利！");
735                     winner = 1;
736                     return true;
737                 } else if (checkPerpetual()) {
738                     System.out.println ("千日手");
739                     System.out.println ("引き分けです");
740                     winner = 0;
741                     return true;
742                 } else {
743                     return false;
744                 }
745             }
746         }
747 
748          /**
749           * 詰みの判定
750           * ライオンの移動可能な手がなければ詰み
751           * @param int playerNum プレイヤー番号
752           * @return 詰みかどうか
753           */
754         public boolean isMate(int playerNum) {
755             return (movableList.size() == 0);
756         }
757 
758         public boolean isChecked(int playerNum) {
759             int file, rank;
760             if(playerNum == 0) {
761                 file = lion.file(); // ライオンの座標
762                 rank = lion.rank();
763                 //自分の駒視点で左から時計回り
764                 //E-犬 1 からの王手
765                 if (board [rank][file-1] == E_DOG1) return true; // 左からの王手
766                 else if (board [rank-1][file-1] == E_DOG1) return true; // 左上から
    の王手
767                 else if (board [rank-1][file] == E_DOG1) return true; // 上からの王
    手
768                 else if (board [rank-1][file+1] == E_DOG1) return true; // 右上から
    の王手
769                 else if (board [rank][file+1] == E_DOG1) return true; // 右からの王
    手
770                 else if (board [rank+1][file] == E_DOG1) return true; // 下からの王
```

```
手
771                 // E-犬 2 からの王手
772                 else if (board [rank][file-1] == E_DOG2) return true; // 左からの王
手
773                 else if (board [rank-1][file-1] == E_DOG2) return true; // 左上から
の王手
774                 else if (board [rank-1][file] == E_DOG2) return true; // 上からの王
手
775                 else if (board [rank-1][file+1] == E_DOG2) return true; // 右上から
の王手
776                 else if (board [rank][file+1] == E_DOG2) return true; // 右からの王
手
777                 else if (board [rank+1][file] == E_DOG2) return true; // 下からの王
手
778                 // E-猫 1 からの王手
779                 else if (board [rank-1][file-1] == E_NEKO1) return true; // 左上か
らの王手
780                 else if (board [rank-1][file] == E_NEKO1) return true; // 上からの
王手
781                 else if (board [rank-1][file+1] == E_NEKO1) return true; // 右上か
らの王手
782                 else if (board [rank+1][file+1] == E_NEKO1) return true; // 右下か
らの王手
783                 else if (board [rank+1][file-1] == E_NEKO1) return true; // 左下か
らの王手
784                 // E-猫 2 からの王手
785                 else if (board [rank-1][file-1] == E_NEKO2) return true; // 左上か
らの王手
786                 else if (board [rank-1][file] == E_NEKO2) return true; // 上からの
王手
787                 else if (board [rank-1][file+1] == E_NEKO2) return true; // 右上か
らの王手
788                 else if (board [rank+1][file+1] == E_NEKO2) return true; // 右下か
らの王手
789                 else if (board [rank+1][file-1] == E_NEKO2) return true; // 左下か
らの王手
790                 // E-成り猫 1 からの王手
791                 else if (board [rank][file-1] == E_CAT1) return true; // 左からの王
手
792                 else if (board [rank-1][file-1] == E_CAT1) return true; // 左上から
の王手
793                 else if (board [rank-1][file] == E_CAT1) return true; // 上からの王
手
794                 else if (board [rank-1][file+1] == E_CAT1) return true; // 右上から
の王手
795                 else if (board [rank][file+1] == E_CAT1) return true; // 右からの王
手
796                 else if (board [rank+1][file] == E_CAT1) return true; // 下からの王
手
797                 // E-成り猫 2 からの王手
798                 else if (board [rank][file-1] == E_CAT2) return true; // 左からの王
手
799                 else if (board [rank-1][file-1] == E_CAT2) return true; // 左上から
```

```
の王手
800                 else if (board [rank-1][file] == E_CAT2) return true; // 上からの王
手
801                 else if (board [rank-1][file+1] == E_CAT2) return true; // 右上から
の王手
802                 else if (board [rank][file+1] == E_CAT2) return true; // 右からの王
手
803                 else if (board [rank+1][file] == E_CAT2) return true; // 下からの王
手
804                 // E-ひよこ 1 からの王手
805                 else if (board [rank-1][file] == E_HIYOKO1) return true; // 上から
の王手
806                 // E-ひよこ 2 からの王手
807                 else if (board [rank-1][file] == E_HIYOKO2) return true; // 上から
の王手
808                 // E-ひよこ 3 からの王手
809                 else if (board [rank-1][file] == E_HIYOKO3) return true; // 上から
の王手
810                 // E-成りひよこ 1 からの王手
811                 else if (board [rank][file-1] == E_BIRD1) return true; // 左からの
王手
812                 else if (board [rank-1][file-1] == E_BIRD1) return true; // 左上か
らの王手
813                 else if (board [rank-1][file] == E_BIRD1) return true; // 上からの
王手
814                 else if (board [rank-1][file+1] == E_BIRD1) return true; // 右上か
らの王手
815                 else if (board [rank][file+1] == E_BIRD1) return true; // 右からの
王手
816                 else if (board [rank+1][file] == E_BIRD1) return true; // 下からの
王手
817                 // E-成りひよこ 2 からの王手
818                 else if (board [rank][file-1] == E_BIRD2) return true; // 左からの
王手
819                 else if (board [rank-1][file-1] == E_BIRD2) return true; // 左上か
らの王手
820                 else if (board [rank-1][file] == E_BIRD2) return true; // 上からの
王手
821                 else if (board [rank-1][file+1] == E_BIRD2) return true; // 右上か
らの王手
822                 else if (board [rank][file+1] == E_BIRD2) return true; // 右からの
王手
823                 else if (board [rank+1][file] == E_BIRD2) return true; // 下からの
王手
824                 // E-成りひよこ 3 からの王手
825                 else if (board [rank][file-1] == E_BIRD3) return true; // 左からの
王手
826                 else if (board [rank-1][file-1] == E_BIRD3) return true; // 左上か
らの王手
827                 else if (board [rank-1][file] == E_BIRD3) return true; // 上からの
王手
828                 else if (board [rank-1][file+1] == E_BIRD3) return true; // 右上か
らの王手
```

```
829                 else if (board [rank][file+1] == E_BIRD3) return true; // 右からの
    王手
830                 else if (board [rank+1][file] == E_BIRD3) return true; // 下からの
    王手
831                 else return false;
832             }else {
833                 file = e_lion.file(); // E-ライオンの座標
834                 rank = e_lion.rank();
835                 //自分の駒視点
836                 //犬 1 からの王手
837                 if (board [rank][file-1] == DOG1) return true; // 左からの王手
838                 else if (board [rank-1][file] == DOG1) return true; // 上からの王手
839                 else if (board [rank][file+1] == DOG1) return true; // 右からの王手
840                 else if (board [rank+1][file+1] == DOG1) return true; // 右下からの
    王手
841                 else if (board [rank+1][file] == DOG1) return true; // 下からの王手
842                 else if (board [rank+1][file-1] == DOG1) return true; // 左下からの
    王手
843                 // 犬 2 からの王手
844                 if (board [rank][file-1] == DOG2) return true; // 左からの王手
845                 else if (board [rank-1][file] == DOG2) return true; // 上からの王手
846                 else if (board [rank][file+1] == DOG2) return true; // 右からの王手
847                 else if (board [rank+1][file+1] == DOG2) return true; // 右下からの
    王手
848                 else if (board [rank+1][file] == DOG2) return true; // 下からの王手
849                 else if (board [rank+1][file-1] == DOG2) return true; // 左下からの
    王手
850                 // 猫 1 からの王手
851                 else if (board [rank-1][file-1] == NEKO1) return true; // 左上から
    の王手
852                 else if (board [rank-1][file] == NEKO1) return true; // 上からの王
    手
853                 else if (board [rank-1][file+1] == NEKO1) return true; // 右上から
    の王手
854                 else if (board [rank+1][file+1] == NEKO1) return true; // 右下から
    の王手
855                 else if (board [rank+1][file-1] == NEKO1) return true; // 左下から
    の王手
856                 // 猫 2 からの王手
857                 else if (board [rank-1][file-1] == NEKO2) return true; // 左上から
    の王手
858                 else if (board [rank-1][file] == NEKO2) return true; // 上からの王
    手
859                 else if (board [rank-1][file+1] == NEKO2) return true; // 右上から
    の王手
860                 else if (board [rank+1][file+1] == NEKO2) return true; // 右下から
    の王手
861                 else if (board [rank+1][file-1] == NEKO2) return true; // 左下から
    の王手
862                 //成り猫 1 からの王手
863                 if (board [rank][file-1] == CAT1) return true; // 左からの王手
864                 else if (board [rank-1][file] == CAT1) return true; // 上からの王手
865                 else if (board [rank][file+1] == CAT1) return true; // 右からの王手
```

```
866             else if (board [rank+1][file+1] == CAT1) return true; // 右下からの
    王手
867             else if (board [rank+1][file] == CAT1) return true; // 下からの王手
868             else if (board [rank+1][file-1] == CAT1) return true; // 左下からの
    王手
869             //成り猫２ からの王手
870             if (board [rank][file-1] == CAT2) return true; // 左からの王手
871             else if (board [rank-1][file] == CAT2) return true; // 上からの王手
872             else if (board [rank][file+1] == CAT2) return true; // 右からの王手
873             else if (board [rank+1][file+1] == CAT2) return true; // 右下からの
    王手
874             else if (board [rank+1][file] == CAT2) return true; // 下からの王手
875             else if (board [rank+1][file-1] == CAT2) return true; // 左下からの
    王手
876             // ひよこ１からの王手
877             else if (board [rank+1][file] == HIYOKO1) return true; // 下からの
    王手
878             // ひよこ２からの王手
879             else if (board [rank+1][file] == HIYOKO2) return true; // 下からの
    王手
880             // ひよこ３からの王手
881             else if (board [rank+1][file] == HIYOKO3) return true; // 下からの
    王手
882             //成りひよこ１ からの王手
883             if (board [rank][file-1] == BIRD1) return true; // 左からの王手
884             else if (board [rank-1][file] == BIRD1) return true; // 上からの王
    手
885             else if (board [rank][file+1] == BIRD1) return true; // 右からの王
    手
886             else if (board [rank+1][file+1] == BIRD1) return true; // 右下から
    の王手
887             else if (board [rank+1][file] == BIRD1) return true; // 下からの王
    手
888             else if (board [rank+1][file-1] == BIRD1) return true; // 左下から
    の王手
889             //成りひよこ２ からの王手
890             if (board [rank][file-1] == BIRD2) return true; // 左からの王手
891             else if (board [rank-1][file] == BIRD2) return true; // 上からの王
    手
892             else if (board [rank][file+1] == BIRD2) return true; // 右からの王
    手
893             else if (board [rank+1][file+1] == BIRD2) return true; // 右下から
    の王手
894             else if (board [rank+1][file] == BIRD2) return true; // 下からの王
    手
895             else if (board [rank+1][file-1] == BIRD2) return true; // 左下から
    の王手
896             //成りひよこ３ からの王手
897             if (board [rank][file-1] == BIRD3) return true; // 左からの王手
898             else if (board [rank-1][file] == BIRD3) return true; // 上からの王
    手
899             else if (board [rank][file+1] == BIRD3) return true; // 右からの王
    手
```

```
900                 else if (board [rank+1][file+1] == BIRD3) return true; // 右下からの王手
901                 else if (board [rank+1][file] == BIRD3) return true; // 下からの王手
902                 else if (board [rank+1][file-1] == BIRD3) return true; // 左下からの王手
903
904                 else return false;
905             }
906         }
907
908         /**
909          * 候補手の作成
910          * movableList に移動可能な手を保存する
911          * @param int playerNum プレイヤー番号
912          */
913         public void createMovableList(int playerNum) {
914             movableList = new ArrayList<>();
915             if(playerNum == 0) { // 先手の場合
916                 if(lion != null) { // 盤面にライオンがある場合
917                     movableList.addAll (lion.movableList (board));  // ライオンが移動可能な手
918                 }
919                 if(dog1 != null) { // 盤面に犬 1 がある場合
920                     movableList.addAll (dog1.movableList(board));  // 犬 1 が移動可能な手
921                 }
922                 if(dog2 != null) { // 盤面に犬 2 がある場合
923                     movableList.addAll (dog2.movableList(board));
924                 }
925                 if(neko1 != null) { // 盤面に猫 1 がある場合
926                     movableList.addAll (neko1.movableList (board));
927                 }
928                 if(neko2 != null) { // 盤面に猫 2 がある場合
929                     movableList.addAll (neko2.movableList (board));
930                 }
931                 if(hiyoko1 != null) { // 盤面にひよこ 1 がある場合
932                     movableList.addAll (hiyoko1.movableList (board));
933                 }
934                 if(hiyoko2 != null) { // 盤面にひよこ 2 がある場合
935                     movableList.addAll (hiyoko2.movableList (board));
936                 }
937                 if(hiyoko3 != null) { // 盤面にひよこ 3 がある場合
938                     movableList.addAll (hiyoko3.movableList (board));
939                 }
940                 if(cat1 != null) { // 盤面に成り猫 1 がある場合
941                     movableList.addAll (cat1.movableList (board));  // 成り猫 1 が移動可能な手
942                 }
943                 if(cat2 != null) { // 盤面に成り猫 2 がある場合
944                     movableList.addAll (cat2.movableList (board));  // 成り猫 2 が移動可能な手
945                 }
```

```
946                 if(bird1 != null) { // 盤面に成りひよこ 1 がある場合
947                     movableList.addAll (bird1.movableList (board));  // 成りひよこ
    1 が移動可能な手
948                 }
949                 if(bird2 != null) { // 盤面に成りひよこ 2 がある場合
950                     movableList.addAll (bird2.movableList (board));  // 成りひよこ
    2 が移動可能な手
951                 }
952                 if(bird3 != null) { // 盤面に成りひよこ 3 がある場合
953                     movableList.addAll (bird3.movableList (board));  // 成りひよこ
    3 が移動可能な手
954                 }
955                 if(isChecked(0)) { // ライオンに王手が掛かっている場合
956                     for(int i = 0; i < movableList.size();) {
957                         Board nextBoard = nextBoard(movableList.get(i), playerNum);
958                         if(nextBoard.isChecked(0)) { // 移動後に王手が掛かっている場
    合
959                             movableList.remove(i); // 移動後で王手にかかっている手を削
    除する
960                         }else {
961                             ++i;
962                         }
963                     }
964                 }
965             }else { // 後手の場合
966                 if(e_lion != null) { // 盤面に E-ライオンがある場合
967                     movableList.addAll (e_lion.movableList (board));  // E-ライオン
    が移動可能な手
968                 }
969                 if(e_dog1 != null) { // 盤面に E-犬 1 がある場合
970                     movableList.addAll (e_dog1.movableList(board));
971                 }
972                 if(e_dog2 != null) { // 盤面に E-犬 2 がある場合
973                     movableList.addAll (e_dog2.movableList(board));
974                 }
975                 if(e_neko1 != null) { // 盤面に E-猫 1 がある場合
976                     movableList.addAll (e_neko1.movableList(board));
977                 }
978                 if(e_neko2 != null) { // 盤面に E-猫 2 がある場合
979                     movableList.addAll (e_neko2.movableList(board));
980                 }
981                 if(e_hiyoko1 != null) { // 盤面に E-ひよこ 1 がある場合
982                     movableList.addAll (e_hiyoko1.movableList(board));
983                 }
984                 if(e_hiyoko2 != null) { // 盤面に E-ひよこ 2 がある場合
985                     movableList.addAll (e_hiyoko2.movableList(board));
986                 }
987                 if(e_hiyoko3 != null) { // 盤面に E-ひよこ 3 がある場合
988                     movableList.addAll (e_hiyoko3.movableList(board));
989                 }
990                 if(e_cat1 != null) { // 盤面に E-成り猫 1 がある場合
991                     movableList.addAll (e_cat1.movableList (board));  // E-成り猫 1
    が移動可能な手
```

```
992                 }
993                 if(e_cat2 != null) { // 盤面に E-成り猫 2 がある場合
994                     movableList.addAll (e_cat2.movableList (board));  // E-成り猫 2
    が移動可能な手
995                 }
996                 if(e_bird1 != null) { // 盤面に E-成りひよこ 1 がある場合
997                     movableList.addAll (e_bird1.movableList (board));  // E-成 り ひ
    よこ 1 が移動可能な手
998                 }
999                 if(e_bird2 != null) { // 盤面に E-成りひよこ 2 がある場合
1000                    movableList.addAll (e_bird2.movableList (board));  // E-成 り ひ
    よこ 2 が移動可能な手
1001                }
1002                if(e_bird3 != null) { // 盤面に E-成りひよこ 3 がある場合
1003                    movableList.addAll (e_bird3.movableList (board));  // E-成 り ひ
    よこ 3 が移動可能な手
1004                }
1005                if(isChecked(1)) { // E-ライオンに王手が掛かっている場合
1006                    for(int i = 0; i < movableList.size();) {
1007                        Board nextBoard = nextBoard(movableList.get(i), playerNum);
1008                        if(nextBoard.isChecked(1)) { // 移動後に王手が掛かっている場
    合
1009                            movableList.remove(i); // 移動後で王手にかかっている手を削
    除する
1010                        }else {
1011                            ++i;
1012                        }
1013                    }
1014                }
1015            }
1016        }
1017
1018        /**
1019         * 現在の盤面の評価値を表示する
1020         * @param int playerNum プレイヤー番号
1021         * @return  評価値
1022         */
1023        public int value(int playerNum) {
1024            checkmate = false;
1025            stalemate = false; //打つ手がないこと
1026            /* 現時点ですでに詰んでいるかどうかのチェック */
1027            if (playerNum == 0) { // A チームの手番
1028                if (lion == null) { // ライオンが取られた
1029                    checkmate = true;
1030                    return Integer.MIN_VALUE; // 評価値無限小
1031                }else if (resign) { // B チームが投了した
1032                    return Integer.MAX_VALUE; // 評価値無限大
1033                }
1034            }else { // B チームの手番
1035                if (e_lion == null) { // E-ライオンが取られた
1036                    checkmate = true;
1037                    return Integer.MAX_VALUE; // 評価値無限大
1038                }else if(resign) { // A チームがリタイアした場合
```

```
                    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;
            value += 15; //ライオンの評価値
            if(dog1 != null) { // 盤面に犬 1 がある場合
                value += 4;
                if(Math.abs(lion.rank() - dog1.rank()) == 1) { // 自ライオンとの距離が 1 のとき
                    value += 2;
                }else if(Math.abs(lion.rank() - dog1.rank()) == 2) { // 自ライオンとの距離が 2 のとき
                    value += 1;
                }else {
                    value += 0;
                }
            }
            if(dog2 != null) { // 盤面に犬 2 がある場合
                value += 4;
                if(Math.abs(lion.rank() - dog2.rank()) == 1) { // 自ライオンとの距離が 1 のとき
                    value += 2;
                }else if(Math.abs(lion.rank() - dog2.rank()) == 2) { // 自ライオンとの距離が 2 のとき
                    value += 1;
                }else {
                    value += 0;
                }
            }
            if(neko1 != null) { // 盤面に猫 1 がある場合
                value += 4;
                if(Math.abs(e_lion.rank() - neko1.rank()) == 1) { // 敵ライオンとの距離が 1 のとき
                    value += 2;
                }else if(Math.abs(e_lion.rank() - neko1.rank()) == 2) {// 敵ライオンとの距離が 2 のとき
                    value += 1;
                }else {
                    value += 0;
                }
            }
            if(neko2 != null) { // 盤面に猫 2 がある場合
                value += 4;
                if(Math.abs(e_lion.rank() - neko2.rank()) == 1) { // 敵ライオンとの距離が 1 のとき
                    value += 2;
                }else if(Math.abs(e_lion.rank() - neko2.rank()) == 2) { // 敵ライオンとの距離が 2 のとき
                    value += 1;
                }else {
                    value += 0;
                }
```

```
1084            }
1085            if(hiyoko1 != null) { // 盤面にひよこ 1 がある場合
1086                value += hiyoko1.rank() - 2; // 敵陣に近いほど高評価
1087            }
1088            if(hiyoko2 != null) { // 盤面にひよこ 2 がある場合
1089                value += hiyoko2.rank() - 2;
1090            }
1091            if(hiyoko3 != null) { // 盤面にひよこ 3 がある場合
1092                value += hiyoko3.rank() - 2;
1093            }
1094            if(cat1 != null) { // 盤面に成り猫 1 がある場合
1095                value += 4;
1096                if(Math.abs(e_lion.rank() - cat1.rank()) == 1) { // 敵ライオンとの距
    離が 1 のとき
1097                    value += 2;
1098                }else if(Math.abs(e_lion.rank() - cat1.rank()) == 2) { // 敵ライオ
    ンとの距離が 2 のとき
1099                    value += 1;
1100                }else {
1101                    value += 0;
1102                }
1103            }
1104            if(cat2 != null) { // 盤面に犬 2 がある場合
1105                value += 4;
1106                if(Math.abs(e_lion.rank() - cat2.rank()) == 1) { // 敵ライオンとの距
    離が 1 のとき
1107                    value += 2;
1108                }else if(Math.abs(e_lion.rank() - cat2.rank()) == 2) { // 敵ライオ
    ンとの距離が 2 のとき
1109                    value += 1;
1110                }else {
1111                    value += 0;
1112                }
1113            }
1114            if(bird1 != null) { // 盤面に犬 1 がある場合
1115                value += 4;
1116                if(Math.abs(e_lion.rank() - bird1.rank()) == 1) { // 敵ライオンとの
    距離が 1 のとき
1117                    value += 2;
1118                }else if(Math.abs(e_lion.rank() - bird1.rank()) == 2) { // 敵ライオ
    ンとの距離が 2 のとき
1119                    value += 1;
1120                }else {
1121                    value += 0;
1122                }
1123            }
1124            if(bird2 != null) { // 盤面に犬 2 がある場合
1125                value += 4;
1126                if(Math.abs(e_lion.rank() - bird2.rank()) == 1) { // 敵ライオンとの
    距離が 1 のとき
1127                    value += 2;
1128                }else if(Math.abs(e_lion.rank() - bird2.rank()) == 2) { // 敵ライオ
    ンとの距離が 1 のとき
```

```
1129                    value += 1;
1130                }else {
1131                    value += 0;
1132                }
1133            }
1134            if(bird3 != null) { // 盤面に犬 1 がある場合
1135                value += 4;
1136                if(Math.abs(e_lion.rank() - bird3.rank()) == 1) { // 敵ライオンとの
    距離が 1 のとき
1137                    value += 2;
1138                }else if(Math.abs(e_lion.rank() - bird3.rank()) == 2) { // 敵ライオ
    ンとの距離が 2 のとき
1139                    value += 1;
1140                }else {
1141                    value += 0;
1142                }
1143            }
1144
1145            value -= 15; // E-ライオンの評価値
1146
1147            if(e_dog1 != null) { // 盤面に E-犬 1 がある場合
1148                value -= 4;
1149                if(Math.abs(e_lion.rank() - e_dog1.rank()) == 1) { // 自ライオンと
    の距離が 1 のとき
1150                    value -= 2;
1151                }else if(Math.abs(e_lion.rank() - e_dog1.rank()) == 2) { // 自ライ
    オンとの距離が 2 のとき
1152                    value -= 1;
1153                }else {
1154                    value += 0;
1155                }
1156            }
1157            if(e_dog2 != null) { // 盤面に E-犬 2 がある場合
1158                value -= 4;
1159                if(Math.abs(e_lion.rank() - e_dog2.rank()) == 1) { // 自ライオンと
    の距離が 1 のとき
1160                    value -= 2;
1161                }else if(Math.abs(e_lion.rank() - e_dog2.rank()) == 2) { // 自ライ
    オンとの距離が 2 のとき
1162                    value -= 1;
1163                }else {
1164                    value += 0;
1165                }
1166            }
1167            if(e_neko1 != null) { // 盤面に E-猫 1 がある場合
1168                value -= 4;
1169                if(Math.abs(lion.rank() - e_neko1.rank()) == 1) { // 敵ライオンとの
    距離が 1 のとき
1170                    value -= 2;
1171                }else if(Math.abs(lion.rank() - e_neko1.rank()) == 2) { // 敵ライオ
    ンとの距離が 2 のとき
1172                    value -= 1;
1173                }else {
```

```
1174                    value -= 0;
1175                }
1176            }
1177            if(e_neko2 != null) { // 盤面に E-猫 2 がある場合
1178                value -= 4;
1179                if(Math.abs(lion.rank() - e_neko2.rank()) == 1) { // 敵ライオンとの
    距離が 1 のとき
1180                    value -= 2;
1181                }else if(Math.abs(lion.rank() - e_neko2.rank()) == 2) { // 敵ライオ
    ンとの距離が 2 のとき
1182                    value -= 1;
1183                }else {
1184                    value -= 0;
1185                }
1186            }
1187            if(e_hiyoko1 != null) { // 盤面に E-ひよこ 1 がある場合
1188                value -= e_hiyoko1.rank() - 5;
1189            }
1190            if(e_hiyoko2 != null) { // 盤面に E-ひよこ 2 がある場合
1191                value -= e_hiyoko2.rank() - 5;
1192            }
1193            if(e_hiyoko3 != null) { // 盤面に E-ひよこ 3 がある場合
1194                value -= e_hiyoko3.rank() - 5;
1195            }
1196            if(e_cat1 != null) { // 盤面に E-成り猫 1 がある場合
1197                value -= 4;
1198                if(Math.abs(lion.rank() - e_cat1.rank()) == 1) {
1199                    value -= 2;
1200                }else if(Math.abs(lion.rank() - e_cat1.rank()) == 2) {
1201                    value -= 1;
1202                }else {
1203                    value -= 0;
1204                }
1205            }
1206            if(e_cat2 != null) { // 盤面に E-成り猫 2 がある場合
1207                value -= 4;
1208                if(Math.abs(lion.rank() - e_cat2.rank()) == 1) { // 敵ライオンとの距
    離が 1 のとき
1209                    value -= 2;
1210                }else if(Math.abs(lion.rank() - e_cat2.rank()) == 2) { // 敵ライオ
    ンとの距離が 2 のとき
1211                    value -= 1;
1212                }else {
1213                    value -= 0;
1214                }
1215            }
1216            if(e_bird1 != null) { // 盤面に E-成りひよこ 1 がある場合
1217                value -= 4;
1218                if(Math.abs(lion.rank() - e_bird1.rank()) == 1) { // 敵ライオンとの
    距離が 1 のとき
1219                    value -= 2;
1220                }else if(Math.abs(lion.rank() - e_bird1.rank()) == 2) { // 敵ライオ
    ンとの距離が 2 のとき
```

```
1221                    value -= 1;
1222                }else {
1223                    value -= 0;
1224                }
1225            }
1226            if(e_bird2 != null) { // 盤面に E-成りひよこ 2 がある場合
1227                value -= 4;
1228                if(Math.abs(lion.rank() - e_bird2.rank()) == 1) { // 敵ライオンとの
    距離が 1 のとき
1229                    value -= 2;
1230                }else if(Math.abs(lion.rank() - e_bird2.rank()) == 2) { // 敵ライオ
    ンとの距離が 2 のとき
1231                    value -= 1;
1232                }else {
1233                    value -= 0;
1234                }
1235            }
1236            if(e_bird3 != null) { // 盤面に E-成りひよこ 2 がある場合
1237                value -= 4;
1238                if(Math.abs(lion.rank() - e_bird3.rank()) == 1) { // 敵ライオンとの
    距離が 1 のとき
1239                    value -= 2;
1240                }else if(Math.abs(lion.rank() - e_bird3.rank()) == 2) { // 敵ライオ
    ンとの距離が 2 のとき
1241                    value -= 1;
1242                }else {
1243                    value -= 0;
1244                }
1245            }
1246            if(playerNum == 0) {
1247                value += movableList.size(); // 移動可能な手が多いほど好評価
1248            }else {
1249                value -= movableList.size(); // 移動可能な手が少ないほど低評価
1250            }
1251            return value;
1252        }
1253
1254        /**
1255         * 現在の盤面の評価値を表示する
1256         * @param int playerNum プレイヤー番号
1257         * @param int depth 先読みする手数
1258         * @return  評価値
1259         */
1260        public int value(int playerNum, int depth) {
1261            if(lion == null) { // ライオンが取られた場合
1262            if(playerNum == 0) { // 先手のとき
1263                    return Integer.MIN_VALUE;
1264            }else { // 後手のとき
1265                    return Integer.MAX_VALUE;
1266                }
1267            }else if(e_lion == null) { // E-ライオンが取られた場合
1268            if(playerNum == 0) { // 先手の場合
1269                    return Integer.MAX_VALUE;
```

```
1270            }else { // 後手の場合
1271                    return Integer.MIN_VALUE;
1272                }
1273            }
1274            createMovableList(playerNum); // 移動可能な手のリストを生成
1275            int value = value (playerNum); // 先読み無しの現在の盤面の評価値を求める
1276            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
1277                return value;
1278            }
1279            // すでに詰んでいる場合はそのまま値を返す
1280            if (checkmate || stalemate || resign) {
1281                return value;
1282            }
1283            int bestValue; // 最も良い評価値
1284            NextMove bestMove = null; // 最も良い手
1285            int nextPlayerNum; // 次の手番プレイヤー
1286            ++lookAheadMoves; // 先読みのために手数を 1 増やす
1287            // A チームの場合
1288            if (playerNum == 0) { // 評価値が高いほど良い手とみなす
1289                nextPlayerNum = 1; // 次の手番プレイヤー
1290                bestValue = Integer.MIN_VALUE; // 盤面の評価値の初期値
1291                for (int i = 0; i < movableList.size(); ++i) {
1292                    NextMove nextMove = movableList.get(i); // i 番目の候補手
1293                    Board nextBoard = nextBoard (nextMove, nextPlayerNum); // 次 の
    盤面を生成する，駒を移動させる
1294                    if(nextBoard.e_lion == null) { // 敵ライオンを取った場合
1295                        value = Integer.MAX_VALUE;
1296                    }else if (nextBoard.checkPerpetual()) { // 千日手の場合
1297                        value = 0;
1298                    }else if(nextBoard.isChecked(nextPlayerNum)) { // 敵 が 王 手 の 場
    合
1299                        value += 6;
1300                        value = nextBoard.value(nextPlayerNum, depth-1);
1301                    }else {
1302                        value = nextBoard.value (nextPlayerNum, depth-1); // 盤 面
    の評価値を計算
1303                    }
1304                    if(value > bestValue) { // 高評価の場合
1305                        bestMove = nextMove; // 高評価の手を保存する
1306                        bestValue = value; // 評価値を代入
1307                    }
1308                    if(bestValue == Integer.MAX_VALUE) { // 評価値が無限大の場合
1309                        --lookAheadMoves; // 先読みが終了し、手数を-1 する
1310                        return Integer.MAX_VALUE;
1311                    }
1312                }
1313            }else { // B チームの場合、低評価の手を良い手とみなす
1314                nextPlayerNum = 0; // 次のプレイヤー番号
1315                bestValue = Integer.MAX_VALUE; // 盤面の評価値の初期値
1316                for (int i=0; i<movableList.size(); ++i) {
1317                    NextMove nextMove = movableList.get(i); // i 番目の候補手
1318                    Board nextBoard = nextBoard (nextMove, nextPlayerNum); // 次 の
    盤面を生成する
```

```
1319                    if(nextBoard.lion == null){ // 敵ライオンを取った場合
1320                        value = Integer.MIN_VALUE;
1321                    }else if (nextBoard.checkPerpetual()) { // 千日手の場合
1322                        value = 0;
1323                    }else if(nextBoard.isChecked(nextPlayerNum)) { // 敵が王手の場
    合
1324                        value -= 6;
1325                        value = nextBoard.value(nextPlayerNum, depth-1);
1326                    }else {
1327                        value = nextBoard.value (nextPlayerNum, depth-1); // 盤 面
    の評価値を計算
1328                    }
1329                    if(value < bestValue) { // 低評価の場合
1330                        bestMove = nextMove; // 低評価の手を保存する
1331                        bestValue = value; // 評価値を代入
1332                    }
1333                    if(bestValue == Integer.MIN_VALUE) { // 評価値が無限小の場合
1334                        --lookAheadMoves; // 先読みが終了し、手数を-1 する
1335                        return Integer.MIN_VALUE;
1336                    }
1337                }
1338            }
1339            --lookAheadMoves; // 先読みが終了したので手数を-1 する
1340            //System.out.println (moves + "+" + lookAheadMoves + ":" + bestMove + "[" + bestValu
1341            return bestValue;
1342        }
1343
1344        public Board nextBoard(NextMove nextMove, int playerNum) {
1345            Board nextBoard = new Board (board);
1346            int movingType = nextMove.type(); // 移動する駒の種類
1347            Piece movingPiece = null; // 移動する駒
1348            // 移動する駒を設定する
1349            switch (movingType) {
1350            case LION :
1351                movingPiece = nextBoard.lion;
1352                break;
1353            case DOG1 :
1354                movingPiece = nextBoard.dog1;
1355                break;
1356            case DOG2 :
1357                movingPiece = nextBoard.dog2;
1358                break;
1359            case NEKO1 :
1360                movingPiece = nextBoard.neko1;
1361                break;
1362            case NEKO2 :
1363                movingPiece = nextBoard.neko2;
1364                break;
1365            case HIYOKO1 :
1366                movingPiece = nextBoard.hiyoko1;
1367                break;
1368            case HIYOKO2 :
1369                movingPiece = nextBoard.hiyoko2;
```

```
1370                 break;
1371             case HIYOKO3 :
1372                 movingPiece = nextBoard.hiyoko3;
1373                 break;
1374             case CAT1 :
1375                 movingPiece = nextBoard.cat1;
1376                 break;
1377             case CAT2 :
1378                 movingPiece = nextBoard.cat2;
1379                 break;
1380             case BIRD1 :
1381                 movingPiece = nextBoard.bird1;
1382                 break;
1383             case BIRD2 :
1384                 movingPiece = nextBoard.bird2;
1385                 break;
1386             case BIRD3 :
1387                 movingPiece = nextBoard.bird3;
1388                 break;
1389             case E_LION :
1390                 movingPiece = nextBoard.e_lion;
1391                 break;
1392             case E_DOG1 :
1393                 movingPiece = nextBoard.e_dog1;
1394                 break;
1395             case E_DOG2 :
1396                 movingPiece = nextBoard.e_dog2;
1397                 break;
1398             case E_NEKO1 :
1399                 movingPiece = nextBoard.e_neko1;
1400                 break;
1401             case E_NEKO2 :
1402                 movingPiece = nextBoard.e_neko2;
1403                 break;
1404             case E_HIYOKO1 :
1405                 movingPiece = nextBoard.e_hiyoko1;
1406                 break;
1407             case E_HIYOKO2 :
1408                 movingPiece = nextBoard.e_hiyoko2;
1409                 break;
1410             case E_HIYOKO3 :
1411                 movingPiece = nextBoard.e_hiyoko3;
1412                 break;
1413             case E_CAT1 :
1414                 movingPiece = nextBoard.e_cat1;
1415                 break;
1416             case E_CAT2 :
1417                 movingPiece = nextBoard.e_cat2;
1418                 break;
1419             case E_BIRD1 :
1420                 movingPiece = nextBoard.e_bird1;
1421                 break;
1422             case E_BIRD2 :
```

```
1423                movingPiece = nextBoard.e_bird2;
1424                break;
1425            case E_BIRD3 :
1426                movingPiece = nextBoard.e_bird3;
1427                break;
1428            }
1429            //int currentFile = movingPiece.file(); // 移動する駒の現在の X 座標
1430            //int currentRank = movingPiece.rank(); // 移動する駒の現在の X 座標
1431            int nextFile = nextMove.nextFile(); // 移動先の X 座標
1432            int nextRank = nextMove.nextRank(); // 移動先の Y 座標
1433            nextBoard.movePiece (movingPiece, movingType, nextFile, nextRank); // 駒
    を移動させる
1434            return nextBoard;
1435        }
1436
1437        /**
1438         * 駒の名前を返す
1439         * @param int type 駒の種類
1440         * @return 駒の名前
1441         */
1442        public String name(int type) {
1443            switch (type) {
1444            case LION :
1445                return "ライオン";
1446            case DOG1 :
1447                return "犬 1";
1448            case DOG2 :
1449                return "犬 2";
1450            case NEKO1 :
1451                return "猫 1";
1452            case NEKO2 :
1453                return "猫 2";
1454            case HIYOKO1 :
1455                return "ひよこ 1";
1456            case HIYOKO2 :
1457                return "ひよこ 2";
1458            case HIYOKO3 :
1459                return "ひよこ 3";
1460            case CAT1 :
1461                return "成り猫 1";
1462            case CAT2 :
1463                return "成り猫 2";
1464            case BIRD1 :
1465                return "成りひよこ 1";
1466            case BIRD2 :
1467                return "成りひよこ 2";
1468            case BIRD3 :
1469                return "成りひよこ 3";
1470            case E_LION :
1471                return "ライオン";
1472            case E_DOG1 :
1473                return "犬 1";
1474            case E_DOG2 :
```

```
1475                return "犬 2";
1476            case E_NEKO1 :
1477                return "猫 1";
1478            case E_NEKO2 :
1479                return "猫 2";
1480            case E_HIYOKO1 :
1481                return "ひよこ 1";
1482            case E_HIYOKO2 :
1483                return "ひよこ 2";
1484            case E_HIYOKO3 :
1485                return "ひよこ 3";
1486            case E_CAT1 :
1487                return "成り猫 1";
1488            case E_CAT2 :
1489                return "成り猫 2";
1490            case E_BIRD1 :
1491                return "成りひよこ 1";
1492            case E_BIRD2 :
1493                return "成りひよこ 2";
1494            case E_BIRD3 :
1495                return "成りひよこ 3";
1496            default :
1497                return "?";
1498            }
1499        }
1500
1501        /**
1502         * 現在の盤面を配列に保存する
1503         */
1504        public void recordBoard() {
1505            if (moves + lookAheadMoves < maxMove) {
1506                for (int r=0; r<6; ++r) {
1507                    for (int f=0; f<5; ++f) {
1508                        boardRecord [moves + lookAheadMoves][r][f] = board[r+1][f+1];
1509                    }
1510                }
1511            }
1512        }
1513
1514        /**
1515         * 千日手になったかどうかの判定
1516         * 条件は同じ手が 3 回出た場合、千日手とみなす
1517         * @return 千日手かどうか
1518         */
1519        boolean checkPerpetual() {
1520            int counter = 0;
1521            for (int i=moves-3; i>=0; i-=2) {
1522                int match = 0;
1523                for (int r=0; r<6; ++r) {
1524                    for (int f=0; f<5; ++f) {
1525                        if (boardRecord [i][r][f] == board[r+1][f+1]) {
1526                            ++match;
1527                        }
```

```
1528                    }
1529                }
1530                if (match == 30) {
1531                    ++counter;
1532                    if (counter >= 3) return true;
1533                }
1534            }
1535            return false;
1536        }
1537
1538
1539        /**
1540         * 手数をリセットする
1541         */
1542        public void resetMoves() {
1543            moves = 0;
1544            lookAheadMoves = 0;
1545        }
1546
1547        /**
1548         * 先読み値の設定
1549         */
1550        public void setMaxDepth(int depth) {
1551            this.maxDepth = depth;
1552        }
1553
1554        /**
1555         * 勝者の番号を返す
1556         * 先手勝ち:1, 後手勝ち:-1, 引き分け:0
1557         * @return  勝者の番号
1558         */
1559        public int winner() {
1560            return winner;
1561        }
1562
1563    }
```

NextMove クラス

以下に第 4.4 章で説明した Board クラスのソースコードを示す.

```
1     package syogi;
2
3   public class NextMove {
4       //  先手の駒  盤面手前
5       final static int LION = 1; // ライオン
6       final static int DOG1 = 2; // 犬 1
7       final static int DOG2 = 3; // 犬  2
8       final static int NEKO1 = 4; // 猫 1
9       final static int NEKO2 = 5; // 猫 2
10      final static int HIYOKO1 = 6; // ひよこ 1
11      final static int HIYOKO2 = 7; // ひよこ 2
12      final static int HIYOKO3 = 8; // ひよこ 3
13
```

```
14      final static int CAT1 = 9; // パワーアップ猫 1
15      final static int CAT2 = 10; // パワーアップ猫 2
16      final static int BIRD1 = 11; // 鶏 1
17      final static int BIRD2 = 12; // 鶏 2
18      final static int BIRD3 = 13; // 鶏 3
19
20      // 後手の駒 盤面奥
21      final static int E_LION = -1; // ライオン
22      final static int E_DOG1 = -2; // 犬 1
23      final static int E_DOG2 = -3; // 犬  2
24      final static int E_NEKO1 = -4; // 猫 1
25      final static int E_NEKO2 = -5; // 猫 2
26      final static int E_HIYOKO1 = -6; // ひよこ 1
27      final static int E_HIYOKO2 = -7; // ひよこ 2
28      final static int E_HIYOKO3 = -8; // ひよこ 3
29
30      final static int E_CAT1 = -9; // パワーアップ猫 1
31      final static int E_CAT2 = -10; // パワーアップ猫 2
32      final static int E_BIRD1 = -11; // 鶏 1
33      final static int E_BIRD2 = -12; // 鶏 2
34      final static int E_BIRD3 = -13; // 鶏 3
35
36      final static boolean isChessStyleScore = false; // 棋譜表記がチェスか将棋かどうか
37
38      int type; // 駒の種類
39      int nextFile; // 移動先の X 座標
40      int nextRank; // 移動先の Y 座標
41      int value; // 移動した場合の盤面の評価値
42
43      /**
44       * コンストラクタ
45       * @param int type 駒の種類
46       * @param int nextFile  移動先の X 座標
47       * @param int nextRank  移動先の Y 座標
48       */
49      public NextMove(int type, int nextFile, int nextRank) {
50          this.type = type;
51          this.nextFile = nextFile;
52          this.nextRank = nextRank;
53      }
54
55      /**
56       * 駒の種類を返す
57       * @return  駒の種類
58       */
59      public int type() {
60          return type;
61      }
62
63      /**
64       * 移動先の X 座標を返す
65       * @return  移動先の X 座標
```

```
66          */
67         public int nextFile() {
68             return nextFile;
69         }
70 
71         /**
72          * 移動先の Y 座標を返す
73          * @return   移動先の Y 座標
74          */
75         public int nextRank() {
76             return nextRank;
77         }
78 
79     }
```

Piece クラス

以下に第 4.5 章で説明した Piece クラスのソースコードを示す.

```
1      package syogi;
2 
3   import java.util.ArrayList;
4 
5   public class Piece {
6       //  先手の駒  盤面手前
7       final static int LION = 1; // ライオン
8       final static int DOG1 = 2; // 犬 1
9       final static int DOG2 = 3; // 犬  2
10      final static int NEKO1 = 4; // 猫 1
11      final static int NEKO2 = 5; // 猫 2
12      final static int HIYOKO1 = 6; // ひよこ 1
13      final static int HIYOKO2 = 7; // ひよこ 2
14      final static int HIYOKO3 = 8; // ひよこ 3
15 
16      final static int CAT1 = 9; // パワーアップ猫 1
17      final static int CAT2 = 10; // パワーアップ猫 2
18      final static int BIRD1 = 11; // 鶏 1
19      final static int BIRD2 = 12; // 鶏 2
20      final static int BIRD3 = 13; // 鶏 3
21 
22      // 後手の駒 盤面奥
23      final static int E_LION = -1; // ライオン
24      final static int E_DOG1 = -2; // 犬 1
25      final static int E_DOG2 = -3; // 犬  2
26      final static int E_NEKO1 = -4; // 猫 1
27      final static int E_NEKO2 = -5; // 猫 2
28      final static int E_HIYOKO1 = -6; // ひよこ 1
29      final static int E_HIYOKO2 = -7; // ひよこ 2
30      final static int E_HIYOKO3 = -8; // ひよこ 3
31 
32      final static int E_CAT1 = -9; // パワーアップ猫 1
33      final static int E_CAT2 = -10; // パワーアップ猫 2
34      final static int E_BIRD1 = -11; // 鶏 1
35      final static int E_BIRD2 = -12; // 鶏 2
```

```
36      final static int E_BIRD3 = -13; // 鶏 3
37
38      final static int EMPTY = 0; // 空白
39      final static int BORDER = Integer.MAX_VALUE; // 盤外
40      boolean isOnBoard; // 駒が盤面上にあるかどうか
41      int x; // x 座標
42      int y; // y 座標
43      int type; // 駒の種類
44      int movableFileVector[], movableRankVector[]; // 移動可能方向
45
46      // 駒が移動可能な X 座標と Y 座標の方向
47      // ライオン{左, 左上, 上, 右上, 右, 右下, 下, 左下}
48      final static int[] lion_X = {-1, -1, 0, 1, 1, 1, 0, -1};
49      final static int[] lion_Y = {0, -1, -1, -1, 0, 1, 1, 1};
50      // 犬 {左、左上、上、右上、右、下}
51      final static int[] dog_X = {-1, -1, 0, 1, 1, 0};
52      final static int[] dog_Y = {0, -1, -1, -1, 0, 1};
53      // ネコ {左上、上、右上、右下、左下}
54      final static int[] neko_X = {-1, 0, 1, 1, -1};
55      final static int[] neko_Y = {-1, -1, -1, 1, 1};
56      // ひよこ {上}
57      final static int[] hiyoko_X = {0};
58      final static int[] hiyoko_Y = {-1};
59
60
61      // 猫（犬と同じ動きになる）
62      final static int[] cat_X = {-1, -1, 0, 1, 1, 0};
63      final static int[] cat_Y = {0, -1, -1, -1, 0, 1};
64      // トリ（犬と同じ動きになる）
65      final static int[] bird_X = {-1, -1, 0, 1, 1, 0};
66      final static int[] bird_Y = {0, -1, -1, -1, 0, 1};
67
68      // 敵駒
69      final static int[] e_lion_X = {-1, -1, 0, 1, 1, 1, 0, -1};
70      final static int[] e_lion_Y = {0, -1, -1, -1, 0, 1, 1, 1};
71      final static int[] e_dog_X = {1, 1, 0, -1, -1, 0};
72      final static int[] e_dog_Y = {0, 1, 1, 1, 0, -1};
73      final static int[] e_neko_X = {1, 0, -1, -1, 1};
74      final static int[] e_neko_Y = {1, 1, 1, -1, -1};
75      final static int[] e_hiyoko_X = {0};
76      final static int[] e_hiyoko_Y = {1};
77
78      // 猫（犬と同じ動きになる）
79      final static int[] e_cat_X = {1, 1, 0, -1, -1, 0};
80      final static int[] e_cat_Y = {0, 1, 1, 1, 0, -1};
81      // トリ（犬と同じ動きになる）
82      final static int[] e_bird_X = {1, 1, 0, -1, -1, 0};
83      final static int[] e_bird_Y = {0, 1, 1, 1, 0, -1};
84
85
86      /**
87       * コンストラクタ
88       * @param int type 駒の種類
```

```
89          */
90         public Piece(int type) {
91             this.type = type;
92             setMovableVector(); // 駒の移動方向を設定
93             setInitialPosition(); // 初期位置を設定
94         }
95 
96         /**
97          * コンストラクタ
98          * @param int type 駒の種類
99          * @param int file 横
100         * @param int rank  縦
101         */
102        public Piece(int type, int file, int rank) {
103            this.type = type;
104            setMovableVector(); // 駒の移動方向を設定
105            this.x = file;
106            this.y = rank;
107            this.isOnBoard = true; // 初期は盤面上にある
108        }
109 
110        /*
111         * 駒の移動可能方向をセットする
112         */
113        public void setMovableVector() {
114            switch(type) { // 駒の種類で分岐する
115            case LION:
116                movableFileVector = lion_X;
117                movableRankVector = lion_Y;
118                break;
119            case DOG1:
120            case DOG2:
121            case CAT1:
122            case CAT2:
123            case BIRD1:
124            case BIRD2:
125            case BIRD3:
126                movableFileVector = dog_X;
127                movableRankVector = dog_Y;
128                break;
129            case NEKO1:
130            case NEKO2:
131                movableFileVector = neko_X;
132                movableRankVector = neko_Y;
133                break;
134            case HIYOKO1:
135            case HIYOKO2:
136            case HIYOKO3:
137                movableFileVector = hiyoko_X;
138                movableRankVector = hiyoko_Y;
139                break;
140            case E_LION:
141                movableFileVector = e_lion_X;
```

```
142             movableRankVector = e_lion_Y;
143             break;
144         case E_DOG1:
145         case E_DOG2:
146         case E_CAT1:
147         case E_CAT2:
148         case E_BIRD1:
149         case E_BIRD2:
150         case E_BIRD3:
151             movableFileVector = e_dog_X;
152             movableRankVector = e_dog_Y;
153             break;
154         case E_NEKO1:
155         case E_NEKO2:
156             movableFileVector = e_neko_X;
157             movableRankVector = e_neko_Y;
158             break;
159         case E_HIYOKO1:
160         case E_HIYOKO2:
161         case E_HIYOKO3:
162             movableFileVector = e_hiyoko_X;
163             movableRankVector = e_hiyoko_Y;
164             break;
165         default:
166             System.out.println("駒の種類を認識できません");
167         }
168     }
169     /*
170      * 駒を初期位置に設定する
171      * 座標は盤面の左上を (x,y)=(1,1) とする
172      */
173     public void setInitialPosition() {
174         switch(type) {
175         case LION:
176             y = 6;
177             x = 3;
178             break;
179         case DOG1:
180             y = 6;
181             x = 2;
182             break;
183         case DOG2:
184             y = 6;
185             x = 4;
186             break;
187         case NEKO1:
188             y = 6;
189             x = 1;
190             break;
191         case NEKO2:
192             y = 6;
193             x = 5;
194             break;
```

```
195             case HIYOKO1:
196                 y = 4;
197                 x = 2;
198                 break;
199             case HIYOKO2:
200                 y = 4;
201                 x = 3;
202                 break;
203             case HIYOKO3:
204                 y = 4;
205                 x = 4;
206                 break;
207             case E_LION:
208                 y = 1;
209                 x = 3;
210                 break;
211             case E_DOG1:
212                 y = 1;
213                 x = 2;
214                 break;
215             case E_DOG2:
216                 y = 1;
217                 x = 4;
218                 break;
219             case E_NEKO1:
220                 y = 1;
221                 x = 1;
222                 break;
223             case E_NEKO2:
224                 y = 1;
225                 x = 5;
226                 break;
227             case E_HIYOKO1:
228                 y = 3;
229                 x = 2;
230                 break;
231             case E_HIYOKO2:
232                 y = 3;
233                 x = 3;
234                 break;
235             case E_HIYOKO3:
236                 y = 3;
237                 x = 4;
238                 break;
239             }
240             isOnBoard = true;
241         }
242         /*
243          * 駒を指定した位置に設定する
244          */
245         public void setPosition(int x, int y) {
246             this.x = x;
247             this.y = y;
```

```
248             isOnBoard = true;
249         }
250         /*
251          * 駒を盤上に置く
252          */
253         public void setOnBoard() {
254             isOnBoard = true;
255         }
256         /*
257          * 駒を盤上から消す
258          */
259         public void removeFromBoard() {
260             isOnBoard = false;
261         }
262         /*
263          * 駒が盤上にあるかどうかを判定する
264          */
265         public boolean isOnBoard() {
266             return isOnBoard;
267         }
268         /*
269          * 駒の X 座標を返す
270          */
271         public int file() {
272             return this.x;
273         }
274         /*
275          * 駒の Y 座標を返す
276          */
277         public int rank() {
278             return this.y;
279         }
280         /*
281          * 駒の種類を返す
282          */
283         public int type() {
284             return this.type;
285         }
286         /*
287          * 駒の名前を返す
288          */
289         public String name() {
290             switch(type) {
291             case LION:
292                 return "ライオン";
293             case DOG1:
294             case DOG2:
295                 return "犬";
296             case NEKO1:
297             case NEKO2:
298                 return "猫";
299             case HIYOKO1:
300             case HIYOKO2:
```

```
301             case HIYOKO3:
302                 return "ひよこ";
303             case CAT1:
304             case CAT2:
305                 return "キャット";
306             case BIRD1:
307             case BIRD2:
308             case BIRD3:
309                 return "鳥";
310             case E_LION:
311                 return "ライオン";
312             case E_DOG1:
313             case E_DOG2:
314                 return "犬";
315             case E_NEKO1:
316             case E_NEKO2:
317                 return "猫";
318             case E_HIYOKO1:
319             case E_HIYOKO2:
320             case E_HIYOKO3:
321                 return "ひよこ";
322             case E_CAT1:
323             case E_CAT2:
324                 return "キャット";
325             case E_BIRD1:
326             case E_BIRD2:
327             case E_BIRD3:
328                 return "鳥";
329             default:
330                 return "?";
331             }
332         }
333         /*
334          * 駒を指定の位置に移動する
335          */
336         public void move(int nextFile, int nextRank) {
337             x = nextFile;
338             y = nextRank;
339         }
340 
341         /*
342          * 駒の移動可能な座標を返す
343          */
344         public ArrayList<NextMove> movableList(int[][] board){
345             ArrayList<NextMove> movableList = new ArrayList<NextMove>();
346             // 移動可能な位置（盤外を含め、7x8 として捉える）
347             boolean[][] isMovable = {{false, false, false, false, false, false, false},
348                                      {false, true, true, true, true, true, false},
349                                      {false, true, true, true, true, true, false},
350                                      {false, true, true, true, true, true, false},
351                                      {false, true, true, true, true, true, false},
352                                      {false, true, true, true, true, true, false},
353                                      {false, true, true, true, true, true, false},
```

```
354                                 {false, false, false, false, false, false, false}};
355         switch(type) { // 駒の種類による移動不可能な位置の判定
356         case LION:
357             for(int r = 1; r <= 6; ++r) {
358                 for(int f = 1; f <= 5; ++f) {
359                     switch(board[r][f]) {
360                     case LION: // 自駒の位置には移動できない
361                     case DOG1:
362                     case DOG2:
363                     case NEKO1:
364                     case NEKO2:
365                     case HIYOKO1:
366                     case HIYOKO2:
367                     case HIYOKO3:
368                     case CAT1:
369                     case CAT2:
370                     case BIRD1:
371                     case BIRD2:
372                     case BIRD3:
373                         isMovable[r][f] = false;
374                         break;
375                     case E_LION: // 敵のライオンに取られる位置に移動不可
376                         isMovable[r][f-1] = false;
377                         isMovable[r+1][f-1] = false;
378                         isMovable[r+1][f] = false;
379                         isMovable[r+1][f+1] = false;
380                         isMovable[r][f+1] = false;
381                         isMovable[r-1][f+1] = false;
382                         isMovable[r-1][f] = false;
383                         isMovable[r-1][f-1] = false;
384                         break;
385                     case E_DOG1:
386                     case E_DOG2:
387                     case E_CAT1:
388                     case E_CAT2:
389                     case E_BIRD1:
390                     case E_BIRD2:
391                     case E_BIRD3:
392                         isMovable[r][f+1] = false;
393                         isMovable[r+1][f+1] = false;
394                         isMovable[r+1][f] = false;
395                         isMovable[r+1][f-1] = false;
396                         isMovable[r][f-1] = false;
397                         isMovable[r-1][f] = false;
398                         break;
399                     case E_NEKO1:
400                     case E_NEKO2:
401                         isMovable[r+1][f+1] = false;
402                         isMovable[r+1][f] = false;
403                         isMovable[r+1][f-1] = false;
404                         isMovable[r-1][f-1] = false;
405                         isMovable[r-1][f+1] = false;
406                         break;
```

```
407                     case E_HIYOKO1:
408                     case E_HIYOKO2:
409                     case E_HIYOKO3:
410                         isMovable[r+1][f] = false;
411                         break;
412                     }
413                 }
414             }
415         case DOG1:
416         case DOG2:
417         case NEKO1:
418         case NEKO2:
419         case HIYOKO1:
420         case HIYOKO2:
421         case HIYOKO3:
422         case CAT1:
423         case CAT2:
424         case BIRD1:
425         case BIRD2:
426         case BIRD3:
427             for(int r = 1; r <= 6; ++r) {
428                 for(int f = 1; f <= 5; ++f) {
429                     switch(board[r][f]) {
430                     case LION: // 自駒の位置には移動できない
431                     case DOG1:
432                     case DOG2:
433                     case NEKO1:
434                     case NEKO2:
435                     case HIYOKO1:
436                     case HIYOKO2:
437                     case HIYOKO3:
438                     case CAT1:
439                     case CAT2:
440                     case BIRD1:
441                     case BIRD2:
442                     case BIRD3:
443                         isMovable[r][f] = false;
444                         break;
445                     }
446                 }
447             }
448             break;
449         case E_LION:
450             for(int r = 1; r <= 6; ++r) {
451                 for(int f = 1; f <= 5; ++f) {
452                     switch(board[r][f]) {
453                     case E_LION: // 自駒の位置には移動できない
454                     case E_DOG1:
455                     case E_DOG2:
456                     case E_NEKO1:
457                     case E_NEKO2:
458                     case E_HIYOKO1:
459                     case E_HIYOKO2:
```

```
460                     case E_HIYOKO3:
461                     case E_CAT1:
462                     case E_CAT2:
463                     case E_BIRD1:
464                     case E_BIRD2:
465                     case E_BIRD3:
466                         isMovable[r][f] = false;
467                         break;
468                     case LION:
469                         isMovable[r][f-1] = false;
470                         isMovable[r-1][f-1] = false;
471                         isMovable[r-1][f] = false;
472                         isMovable[r-1][f+1] = false;
473                         isMovable[r][f+1] = false;
474                         isMovable[r+1][f+1] = false;
475                         isMovable[r+1][f] = false;
476                         isMovable[r+1][f-1] = false;
477                         break;
478                     case DOG1:
479                     case DOG2:
480                     case CAT1:
481                     case CAT2:
482                     case BIRD1:
483                     case BIRD2:
484                     case BIRD3:
485                         isMovable[r][f-1] = false;
486                         isMovable[r-1][f-1] = false;
487                         isMovable[r-1][f] = false;
488                         isMovable[r-1][f+1] = false;
489                         isMovable[r][f+1] = false;
490                         isMovable[r-1][f+1] = false;
491                         isMovable[r+1][f] = false;
492                         break;
493                     case NEKO1:
494                     case NEKO2:
495                         isMovable[r-1][f-1] = false;
496                         isMovable[r-1][f] = false;
497                         isMovable[r-1][f+1] = false;
498                         isMovable[r+1][f+1] = false;
499                         isMovable[r+1][f-1] = false;
500                     case HIYOKO1:
501                     case HIYOKO2:
502                     case HIYOKO3:
503                         isMovable[r-1][f] = false;
504                         break;
505                     }
506                 }
507             }
508         case E_DOG1:
509         case E_DOG2:
510         case E_NEKO1:
511         case E_NEKO2:
512         case E_HIYOKO1:
```

```
513             case E_HIYOKO2:
514             case E_HIYOKO3:
515             case E_CAT1:
516             case E_CAT2:
517             case E_BIRD1:
518             case E_BIRD2:
519             case E_BIRD3:
520                 for(int r = 1; r <= 6; ++r) {
521                     for(int f = 1; f <= 5; ++f) {
522                         switch(board[r][f]) {
523                         case E_LION: // 自駒の位置には移動できない
524                         case E_DOG1:
525                         case E_DOG2:
526                         case E_NEKO1:
527                         case E_NEKO2:
528                         case E_HIYOKO1:
529                         case E_HIYOKO2:
530                         case E_HIYOKO3:
531                         case E_CAT1:
532                         case E_CAT2:
533                         case E_BIRD1:
534                         case E_BIRD2:
535                         case E_BIRD3:
536                             isMovable[r][f] = false;
537                             break;
538                         }
539                     }
540                 }
541                 break;
542             }
543             for(int i = 0; i < movableFileVector.length; ++i) { // 移動可能かチェッ
    クする
544                 int nextFile = x + movableFileVector[i];
545                 int nextRank = y + movableRankVector[i];
546                 if(isMovable[nextRank][nextFile]) {
547                     NextMove nextMove = new NextMove(type, nextFile, nextRank); // 移
    動可能リストに代入
548                     movableList.add(nextMove);
549                 }
550             }
551 
552             return movableList;
553         }
554 
555 
556     }
```